

An Assurance-based Approach to Verification and Validation of Human–Robot Teams

Matt Webster,^{*} David Western,[†] Dejanira Araiza-Illan,[‡]
Clare Dixon,^{*} Kerstin Eder,^{†‡} Michael Fisher,^{*} and
Anthony G. Pipe,^{‡§}

Abstract

We present an approach for the verification and validation (V&V) of robot assistants in the context of human-robot interactions (HRI), to demonstrate their trustworthiness through integral assurances on their safety and functional correctness. Trust in robot assistants will allow them to transition from the laboratory into our everyday lives. The complex and unpredictable nature of the real world in which assistant and service robots operate, the limitations on available V&V techniques when used individually, and the consequent lack of confidence on the verification results (or assurances), present challenges to overcome. Our approach, called *assurance-based verification*, addresses these challenges by combining formal verification (model checking), simulation-based testing, and user validation in experiments with a real robot. We demonstrate our assurance-based V&V approach through a handover task, the most critical part of a complex cooperative manufacturing scenario, for which we proposed some safety and liveness requirements to verify and validate. We construct formal models, simulations and an experimental test rig for the HRI. To capture requirements we use temporal logic properties, assertion checkers and informal textual descriptions. This combination of approaches allows V&V of the HRI task at different levels of modelling detail and thoroughness of exploration, thus overcoming the individual limitations of each method. Should the resulting assurances present discrepancies, an iterative process between the three V&V techniques takes place until confidence in these assurances is gained from refining and improving the assets (i.e., system and requirement models) to represent the HRI task in a more truthful manner.

1 Introduction

Robotic assistants that interact with people in an informal, unstructured, and complex manner are increasingly being considered for industrial and domestic do-

^{*}Department of Computer Science, University of Liverpool, Liverpool, UK

[†]Department of Computer Science, University of Bristol, Bristol, UK

[‡]Bristol Robotics Laboratory, Bristol, UK

[§]Faculty of Environment and Technology, University of the West of England, Bristol, UK

mains. In manufacturing, the drive towards more flexible production, quality and consistency in the production, and the reduction of tiring and dangerous tasks, requires that humans work near robots, or even teach and physically interact with them as co-workers.

A way to enhance robots, to allow their safe and trustworthy participation in human–robot interactions (HRI), is the incorporation of safety and fault-recovery mechanisms at all levels (Alami et al., 2006): lower (mechanical, basic controllers), and higher (high-level, decision-making actions). For example, restricting motion when near humans has been applied as a low-level safety solution (Pedrocchi et al., 2013). However, to allow robot assistants to transition from research laboratories and very limited application scenarios such as surveillance, transport or entertainment, to the broader domestic and industrial domains, they need to be demonstrably trustworthy (Eder et al., 2014). Collaborative robots will also need to conform to newly developed standards (ISO 15066, 2016). Thus, HRI requires the development of coherent and credible frameworks for verification and validation (V&V).

A major challenge in V&V of robot assistants is that no single technique is adequate to cover the whole system in practice. “Correct” functioning in an HRI scenario is likely to depend on precise physical details as well as complex high-level interactions. Individually, formal methods such as model checking and theorem proving, simulation-based testing, or experiments in real-world scenarios, cannot examine the entire state space of the interaction with realistic detail. The advantages of these techniques, formal, simulation and experiments, in terms of *coverability* (i.e., the exploration of the state space such as combinations of human–robot actions, or motion ranges) and *realism* can be exploited when combining them.

Combining V&V techniques in the HRI domain yields an additional benefit – trust in the correctness of V&V results. When using a single V&V technique this is hard to achieve. System models used in formal methods or simulation-based testing, and requirements models, are subject to manual input errors, despite efforts in automating translations between models, and translations from code to models. The use of complementary V&V techniques can highlight discrepancies and help system developers gain confidence in the resulting assurances about safety and liveness requirements.

Our contribution, and the approach explored in this paper, combines formal methods, simulation-based testing, and user validation through experiments with a real robot, in the context of HRI. We refer to this approach as *assurance-based verification*, because we seek to gain confidence in the resulting assurances from the complementary V&V techniques. If the assurances agree when verifying and validating the same requirement through the three techniques, we will be more confident in the results. Otherwise, an iterative process is needed to refine and improve the truthfulness of the *assets*, the system and requirement models underpinning each technique. Hence, assurance-based verification provides increased confidence in the assurances, compared to using V&V techniques in isolation. At the same time, by enabling V&V to span across multiple levels of detail or abstraction, our approach provides a thorough exploration of the robot’s range of

behaviours, thus overcoming limitations of individual V&V techniques.

The proposed approach is exemplified through an object handover task, the most critical component of a cooperative manufacture scenario, for the BERT 2 robot (Lenz et al., 2010). We formulated safety and liveness requirements based on relevant standards. We then used this case study to show that an assurance-based verification can provide a higher degree of confidence in the resulting assurances than when using verification techniques in isolation. The instantiation of our approach for the case study comprises, as V&V techniques, probabilistic model checking in PRISM (Kwiatkowska et al., 2011), simulation-based testing in ROS and Gazebo, and an experimental setup in the Bristol Robotics Laboratory (BRL). Relevant assets were developed with these tools in mind. Experiments were designed and conducted at the BRL. A formal model comprising Probabilistic Timed Automata (PTA) was constructed by hand, representing the HRI. Probabilistic Computation Tree Logic (PCTL) (Kwiatkowska et al., 2011) properties were derived from the requirements, to be verified against the formal model. We developed a simulator in ROS–Gazebo, with the real code for the robot and a simulated human coworker. Tests were derived from model-based and pseudorandom techniques, as in our previous work in (Araiza-Illan et al., 2015; Araiza-Illan et al., 2016), to stimulate the HRI components towards checking the satisfaction of the requirements. Automated checkers implemented as assertion monitors, as described in (Araiza-Illan et al., 2015; Araiza-Illan et al., 2016), were also derived from the requirements and added into the simulator. Applying the complementary V&V techniques exposed discrepancies in the resulting assurances, allowing the assets to be examined and refined. Iterating over this process led to agreement between the three techniques, thus providing greater confidence in the correctness of the resulting assurances and the suitability of subsequent design recommendations.

The paper proceeds as follows. Related work is discussed in Section 2. Section 3 presents our assurance-based verification approach, outlining the V&V techniques, their corresponding assets to be developed from the HRI system and its requirements, and their interactions to gain confidence in the resulting assurances. We then introduce the case study, the handover task, and the requirements to be verified in Section 4. Next, we present the instantiation of the proposed assurance-based verification approach for the case study in Section 5, including the development of assets comprising the formal model, the simulator, and the translations of the requirements into logical properties and assertions. We present the V&V results for two of the proposed requirements in Section 6, describing in detail the encountered assurance discrepancies, with the consequent asset refinement and improvement processes until reaching a high degree of confidence in the results. In Section 7 we then demonstrate verification of the remaining requirements using the three verification methods. Section 8 discusses the findings and limitations in the application of the assurance-based verification approach to our case study. Finally, we offer conclusions and directions for future work in Section 9.

2 Related Work

The use of a single V&V method for verification implies a compromise between examining the full state space of a system (in this case, of an HRI), and modelling the system in satisfactory detail. Model checking (Clarke et al., 1999; Fisher, 2011), a formal method, is exhaustive over the state space of a model, but requires abstraction of the full system (e.g., the high-level control algorithms, low-level control and mechanical behaviour, and the code that runs in the robot) into a finite number of states. For this reason, formal verification can be applied to the analysis of high-level decision-making engines for safety and liveness purposes, exemplified by our previous work in HRI scenarios (Bordini et al., 2009; Webster et al., 2015). Reasoning and high-level control algorithms have been verified through model checking for other kinds of autonomous robots such as unmanned aircraft (Webster et al., 2013) and multi-robot swarm systems (Dixon et al., 2012; Konur et al., 2012). Theorem proving, an alternative formal method, has also been used to verify some of the control code of an autonomous robot (Walter et al., 2010) and multi-robot swarms (Behdenna et al., 2009), highlighting the same modelling challenges in terms of abstractions versus accuracy and detail as in model checking.

Unlike formal methods, simulation-based testing is not exhaustive, nor does it offer proof of requirement satisfaction. However, simulators allow more detailed modelling of the physical and low-level implementation aspects (e.g., sensors or joint controllers in the actuators), and the robot’s actual control code can be executed for verification purposes. Although formal models can be run in simulation-mode when model checking is not practical (Nielsen, 2014), dedicated simulators are preferred for testing robotics. For example, a simulator was built in MATLAB in (Kirwan et al., 2013), whereas Arnold and Alexander (2013) used the Player/Stage 2D simulator and Pinho et al. (2014) used the SimTwo simulator in combination with the Robot Operating System (ROS)¹ robot software development framework, to test autonomous navigation control algorithms. Many other simulation and development frameworks, such as ROS, are widely available. In our previous work, we developed simulators in a 3D physical engine, Gazebo², containing models of the robot’s joints and continuous motion in space, along with its continuous environment containing objects and humans (Araiza-Illan et al., 2015; Araiza-Illan et al., 2016).

In other domains, such as microelectronics, both formal methods and simulation-based testing are used in many Electronic Design Automation tools. Simulation and formal methods have been used in combination, to overcome the limitations of model-checking, or to provide human-readable evidence of failures that can be observed at runtime. Emulating these principles, academic formal analysis tools also offer both model checking and simulation-based testing, such as UP-

¹<http://www.ros.org/>

²<http://gazebo.org/>

PAAL (Nielsen, 2014), Event-B³, and the FDR2 tool⁴. Nonetheless, performing both model checking and testing over the same formal model is disadvantageous when gaining confidence in the resulting assurances, as these two V&V processes are subject to the same modelling and coding errors. This problem is highlighted by Kirwan et al. (2013), as they crafted a simulator (in MATLAB) and a formal model (in Promela for the SPIN model checker) of their robot’s software (an autonomous navigation closed-loop system) independently to overcome the limitations of simulations and model checking and gain confidence in their results. Intana et al. (2013) combined the advantages of simulation and formal verification for wireless sensor networks. Simulations in an environment called MiXiM allowed discovering functional issues in a high fidelity model, whereas formal verification in Event-B was used to provide proofs of requirement satisfaction or violation, to strengthen the discoveries during simulation. They do not consider a course of action if the results from simulation contradict the ones from formal verification.

Experiments in real-world scenarios are costly when compared to simulation and formal methods, and cannot thoroughly explore the full state space of an HRI scenario. Simulation and experimentation can be combined through hybrids of human-in-the-loop and simulation, or robot-in-the-loop and simulation as proposed by Petters et al. (2008). Our verification approach combines three techniques: formal methods, simulation-based testing, and experiments, eliminating the need to choose between examining the full state space of an HRI, and modelling the HRI in satisfactory detail.

Our assurance-based verification approach draws on multiple forms of evidence to support a claim. The clear presentation of such arguments, e.g., by Goal Structuring Notation (Kelly and Weaver, 2004), is an important consideration in safety critical systems. Hawkins et al. (2011) describe the importance of separating a safety argument from its accompanying confidence argument, which justifies the sufficiency of confidence in the safety argument. Similar to our approach, but more limited in terms of variety of V&V techniques, the claims computed with a new variant of formal analysis, based on models of flows instead of models of states, are validated by experiments in the lab (Lyons et al., 2013). They applied the verification technique to autonomous navigation algorithms for multi-robot missions for Pioneer-3AT robots, but the validation stage only involved one robot. An approach to test robotic software through co-simulation was presented in (Broenink et al., 2010), where formal verification was used to find deadlocks through the FDR2 tool, whereas models of the robot’s software and hardware at different levels of abstraction allowed a thorough testing of the discrete and continuous interacting components. These multiple simulators run in a synchronized manner in a co-simulation. They do not consider a course of action when finding discrepancies between the formal analysis and the simulations.

In assurance-based verification, we seek agreement between multiple verifica-

³<http://www.event-b.org/>

⁴<http://www.fsel.com/software.html>

tion techniques on a set of equivalent assurances. Discrepancies may arise due to inaccuracies or errors in one or more of the system models, requirements models, or tools used. In several previous works, methods have been proposed for improving the models. For example, formal models are refined iteratively if they produce a spurious property violation after model checking in Counter Example-Guided Abstraction Refinement (CEGAR) (Clarke et al., 2000). An initial detailed model is abstracted to form a simpler upper-approximation for which model checking is tractable. After encountering a violation of a requirement, that model is iteratively and automatically refined to determine whether the violation is spurious (i.e. does not occur in the more detailed model). The level of detail that may be accounted for by such techniques remains limited to that which can be formally modelled. For a system’s software, this may extend to the concrete code design, but for complex cyber-physical systems such as HRI, there will typically be important details that cannot be adequately represented. Our assurance-based verification may be seen as an approach in the spirit of CEGAR, with greater dependence on human judgement to extend beyond formal modelling and accommodate system models between which absolute agreement may not be achievable.

Many approaches have been proposed to verify and validate requirement models, with respect to consistency, completeness, and precision. For example, Heitmeyer (2007) developed a tool that performs formal verification (both model checking and theorem proving) as well as simulation and even code generation, by integrating multiple external tools. Nonetheless, further advantages can be gained when multiple and independently applied V&V techniques are combined to gain confidence in their results, as we propose here.

Frameworks to verify and validate models for simulation tasks, with respect to accuracy and validity, have been proposed in (Robinson, 1997; Sargent, 2013). These models are developed from gathering real-world data, and their V&V continues throughout its simulation use. Dimensions that can be verified are: concept (aspects to be included in the model, such as variables of importance), data (e.g. accuracy, format), timing, control and information flows, and even the code against bugs. Techniques that can be applied for V&V include animation, comparison against other models, and testing (e.g. stress, sensitivity, historical data comparison). Nonetheless, the authors do not prescribe a methodology with associated tools to achieve model V&V.

For ROS-based systems, the accuracy of a formal model with respect to the robot’s control code may be more rigorously examined by standardising the formal description of common ROS components, e.g. using the “ROS graph” formalisation developed in Aitken et al. (2014) to enable automated reconfiguration of ROS systems. Recently, this formalisation has been adopted by Hazim et al. (2016) to apply model-checking to the verification of timing properties of ROS processes. Their approach shows promise for ensuring that the formal model is representative of the system’s performance. Further work is needed to demonstrate whether it can be usefully extended to capture inaccuracies in modelling the environment or the requirements, which may be more challenging to model when considering physical

aspects of the system besides timing.

For both requirement and system models, an approach to ensuring consistency is to generate one model from another by a trusted method. Automated tools can help in this process, such as translations from MATLAB/Simulink control models or control code into formal models for model checking (Xie et al., 2004; Meenakshi et al., 2006). Formal system models can be automatically extracted from real code, as in (Corbett et al., 2000; Gallardo et al., 2012; Mukhopadhyay, 2015), although not many tools are compatible with Python, a popular language for prototyping robotics code. Logical properties may be automatically converted into automata (Gastin and Oddoux, 2001), which may then be encoded as a monitor in the form of a finite-state machine. Huang et al. (2014a) introduce *rosmop*, a tool to automatically convert logical properties into monitors for runtime verification of ROS code. Similar approaches could potentially be adopted in combination with assurance-based verification for HRI to increase the level of confidence in the results, although errors could still propagate when transforming one model into another, e.g., inaccuracies in a logical property will propagate to a monitor.

A simulation-based testing process can be improved by using tests that not only stimulate the system, but can also find faults, using so-called mutation-based test generation (Huang et al., 2014b). A system’s safety and liveness requirements model, crucial in a V&V task, can also be verified for consistency, correctness and completeness, e.g. using a combination of formal methods, static analysis and simulation as in Heitmeyer (2007). If a system is to be designed and implemented from a requirements model, certified code generators (Naks et al., 2009) and code synthesis (e.g., refinement) (Ringert et al., 2014) can be employed. However, the validity of the resulting code is dependent on the accurate representation of non-software aspects of the system in the original model, which is especially challenging in the HRI domain. Furthermore, in practice robots are commonly designed and built by different interacting teams, due to the complexity of the applications.

In summary, various techniques exist to promote correctness in modelling, and to bridge different levels of abstraction for V&V in robotics and other domains. However, none of these spans the full range of realism and coverability needed to thoroughly verify and validate an HRI system, while systematically addressing the possibility for errors to be introduced at any level of abstraction. Confidence in the results of these techniques, if used in isolation, is thus limited. Our proposed approach does not prescribe specific V&V tools and techniques to be used. Automatic translation and connections between the used V&V techniques can be added to the approach, with discretion, to improve confidence or efficiency in the V&V exercise. For instance, in our demonstration we exploit model-based test generation and the ROS-Gazebo compatibility as additional links between simulation-based testing and formal methods.

3 Assurance-Based Verification Approach

As noted in the introduction, assurance-based verification provides increased confidence in the assurances, whilst providing a thorough exploration of the robot’s range of behaviours across multiple levels of abstraction, thus overcoming limitations of individual V&V techniques. Our approach to the V&V of human–robot teams is shown in Figure 1. We propose the combined use of three techniques to verify and validate robots in HRI tasks, which are shown in ellipses. Each method is underpinned by two *assets*: a requirements model, shown in a rectangle, and a system model, shown in an octagon. We introduce the techniques, the assets and the assurance-based verification workflows, indicated by the arrows in Figure 1, in the following subsections.

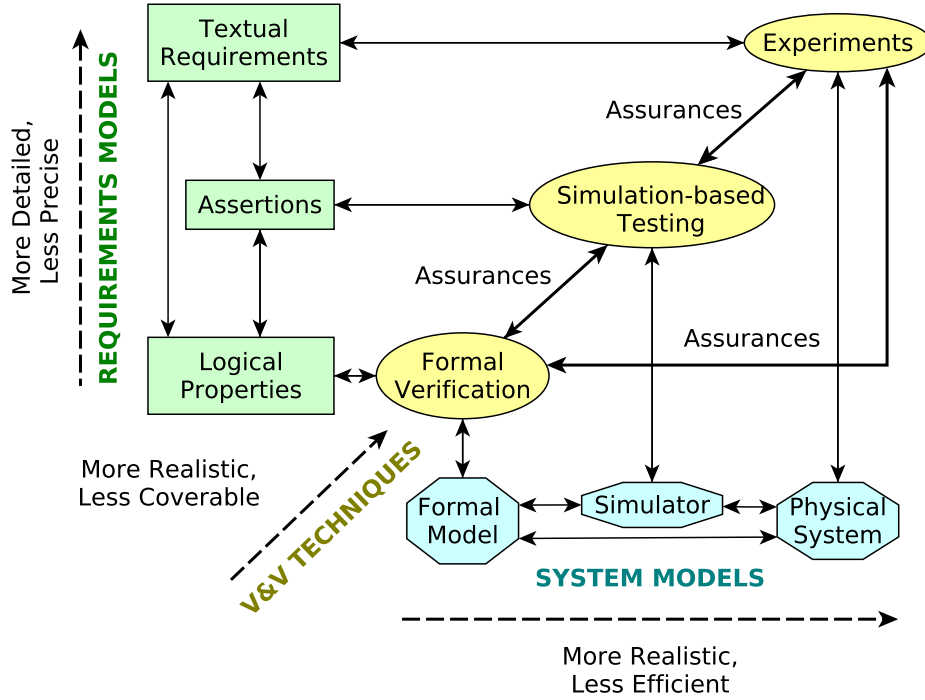


Figure 1: A framework for assurance-based V&V.

3.1 V&V Techniques

Formal verification encapsulates a set of mathematical techniques which are used to prove properties about a *formal model* of a system. Some of the most common formal verification techniques are model checking (Clarke et al., 1999) and theorem proving (Fitting, 1996). In this paper we use model checking, which lets

us verify that formal models (which represent the robot and its non-deterministic environment) satisfy temporal logical properties (derived from requirements) for every possible way in which the models can be executed. As we examine every possible execution of the formal model, we can demonstrate whether or not the model satisfies the temporal logical properties.

In “traditional” model checking, finite state machines are modelled and explored exhaustively in order to determine whether some property holds (Clarke et al., 1999). Properties are typically expressed as logical formulae written in a logical language, e.g., linear-time temporal logic (LTL) or computation-tree logic (CTL). The output of a model checker is typically a Boolean value, true or false, indicating whether the model satisfies a given property. In the case where the model does not satisfy the property, an “error trace” or counter-example is output describing the sequence of states which led to the violation of the property (Fisher, 2011). Probabilistic model checking, explained further in Section 5.1, extends this method to allow the computation of the probability that a given property will be satisfied.

Simulation-based testing involves running a *simulator* under different inputs (or tests), to observe the resulting outputs and determine if the simulated system behaves as intended. Software and hardware components can be modelled to achieve an appropriate compromise between realism, modelling effort, and computational cost, and real code can be run. Nonetheless, the exploration of a system under test is not exhaustive. Systematic methodologies to explore the system under test, such as Coverage-Driven Verification (CDV) (Araiza-Illan et al., 2015; Araiza-Illan et al., 2016), should be used to increase efficiency and effectiveness under computational constraints. A CDV testing process needs testbench components, including a test generator and a driver, to stimulate the system under test, a coverage collector, to keep track of the V&V progress, and a checker modelling the requirements and automating the checking (Piziali, 2004).

Experiments are performed within a real-world *physical system* to verify and validate robots interacting in realistic environments with respect to textual requirements. As experiments often involve human volunteers, health and safety assessments and expensive equipment, the number of times that a particular scenario can be examined is often severely limited when compared to simulation or formal verification. In this paper, experiments are focused towards achieving clear assurances on the principal requirements, as well as to ground the assurance-based verification process in reality.

The diagonal axis in Figure 1 arranges the three techniques based on how realistic and how *coverable* they are. V&V techniques that allow more realistic modelling of a system tend to be less suitable for covering the full state-space of the system, such as it is the case for simulation-based testing. Simulation-based testing can be more realistic than formal verification, as it can account for physical detail that is difficult to model formally, such as the dynamic model of a robot. However, simulation-based testing is less coverable: it is not possible to simulate every conceivable scenario, so a sample is chosen instead. Formal verification is, by definition, exhaustive (100% coverable) (Clarke et al., 1999). Physical experi-

ments are more realistic than simulation-based testing, but are also less coverable because physical experiments are more strictly constrained by time and other resources. The number of simulations that can be conducted in practice will typically be much greater than the number of physical experiments.

3.2 V&V Assets

In Figure 1 it is shown that requirements can be modelled in a number of ways. *Textual requirements* are the written requirements that describe the desired behaviour of a robot, and can also include some assumptions about the human user’s behaviour and the environment in which the robot operates (e.g. all materials required to complete the task are available at the start). Textual requirements are used in experiments to determine whether the robot (i.e., the physical system) satisfies them. Textual requirements for robots are typically based on the needs of the system’s users, but are increasingly based on legal or ethical frameworks specified by a regulatory or standards authority. For example, ISO 15066 (2016) defines many safety requirements for collaborative robots. In practice, verifying a textual requirement in experiments may necessitate refinement of the text with consideration for the actual scenario, to avoid ambiguities.

Assertions are requirements of a system expressed in an assertion specification language using the syntax of programming languages such as C or Python (Foster et al., 2004), or as assertion monitors, such as the ones implemented in (Huang et al., 2014a; Araiza-Illan et al., 2015; Araiza-Illan et al., 2016). Assertions are commonly formulated in a precondition-implies-postcondition manner, and can be implemented directly in the code under testing, or within the simulation models. Tools are available to convert temporal logical properties into monitors for runtime verification, as in (Havelund and Rosu, 2002; Huang et al., 2014a), the latter for testing robots. The systems under verification are stimulated to attempt triggering the preconditions in the assertions, and consequently a check of their respective postconditions. The outcomes of these checks are interpreted to determine if the requirements are satisfied.

A computer-based *simulator*, a program written in a high-level language, contains models of the robot’s behaviour as well as its environment. In simulation-based testing, the simulator program is executed multiple times (computation time allowing), to collect information from the assertion checks and the simulation itself. As mentioned in the introduction, a number of both open-source and proprietary simulation and development frameworks exists in robotics, such as ROS, Player/Stage, Gazebo, V-REP, Webots and others.

Logical properties are logical statements, each of which captures one or more requirements of the system using some formal logic. Different logics can be used for different applications; e.g., if we want to capture requirements relating to time we might use Linear Temporal Logic (LTL) (Fisher, 2011). Alternatively, if we are interested in the probability of the requirement being met then we may use probabilistic computation tree logic (PCTL). Formal modelling tools specialize in

supporting particular types of formal models and temporal logics, such as PCTL by PRISM (Kwiatkowska et al., 2011). *Formal models* are discrete computational descriptions of high-level behaviours. Finite State Automata (FSA) (Clarke et al., 1999) and Probabilistic Timed Automata (PTA) (Parker, 2016) are two examples.

Figure 1 arranges the requirements models in order of how detailed or precise they are. “Detail” here refers to the breadth of realistic detail that could be referred to in the requirement model, while “precision” refers to how specific the expressions may be. A single requirement may be implemented as assertions in many ways, e.g., according to interpretations by different programmers. As assertions are based on programming languages whose semantics are more well-defined than natural languages, we consider assertions to be more precise than textual requirements. Logical properties are, in turn, more precise than assertions and textual requirements, as they have precise, mathematical definitions. On the other hand, assertions can be more detailed than the logical properties, as they can capture aspects of the system which are difficult to specify at higher levels of abstraction (e.g., physical states that depend on modelled dynamics). However the assertions are less detailed than the textual requirements: subjective requirements such as user satisfaction are difficult to model in programming languages. It should also be noted that the more realistic levels of the framework can support a broader set of requirements, since they allow the monitoring of parameters or analysis of components that might not be available in more abstract models.

Ideally, the assets mentioned before would be generated during the development of the robot itself. For example, textual requirements would be developed at the start of the traditional product engineering life cycle, and may be based on standards and regulations such as ISO 10218 (2011) for industrial robots. At the next stage in the life cycle the product would be designed. Simulation and formal analysis are often used in the hardware and software domains at the design stage in order to gain confidence in the correctness of the design with respect to the specifications. Hence formal models and simulators would be developed. This practice can be adopted for the design of robot assistants, as demonstrated by Kirwan et al. (2013) for autonomous navigation. Experiments would be performed during implementation of the robot system in real life HRI, after designing the corresponding setup.

Re-examining equivalent requirements implemented at different abstraction levels of the framework provides an opportunity to refine individual assets to represent the HRI more accurately and truthfully, making the framework robust with respect to human error and providing a high degree of confidence in the resulting assurances. When refining the assets, complexity needs to be carefully managed, e.g. through abstraction. Re-modelling formal models and simulators can result in a state-space explosion and a significant increase in time and memory (Clarke et al., 2012). As explained previously, each level of our framework represents a different compromise between realism and the coverability of the state space. Any decisions affecting the balance of this compromise should be made by those conducting the verification.

If it is not possible to develop all assets during the initial development of the human–robot system — e.g., if the human–robot system has already been developed — the approach can still be applied. In this case, it is necessary to develop assets based on existing materials.

The bidirectional arrows between the different system models in Figure 1, and between the different requirements models, indicate that the development of any of these models may be informed by the equivalent model at another level of abstraction. Such development may be carried out manually or using some of the techniques mentioned in Section 2. Our framework allows for the incorporation of such techniques to suit the application in question.

3.3 Assurance-Based Workflows

Our approach leaves the order in which the different techniques in Figure 1 should be used open. Such decisions should be made with consideration for the specific HRI application in question. Furthermore, these decisions will typically be made in a reactive manner, because insights gained from any of the techniques can lead to modifications in any of the system models or requirements models, necessitating a further stage of V&V to increase confidence in the results, possibly with a different technique.

For example, we could start with a set of logical properties and a formal model of the robot system. Formal verification would then be used to verify that the formal model satisfies the logical properties. This process is indicated by the arrows from “Logical Properties” and “Formal Model” to “Formal Verification” in Figure 1. The result of formal verification is a set of *assurances* that the formal model is correct with respect to the logical properties. During this verification, we may discover that the formal model does not satisfy a particular property. If we trust this verification result, modifications to the formal model may be an appropriate way to explore possible design modifications. The “Simulator” and “Physical System” would then need to be updated accordingly, as represented by the bi-directional arrows between system models in Figure 1. Alternatively, the property violation may be due to an error in the model or in the logical property (i.e., we have incorrectly formalised a requirement). Otherwise, we may wish to manually revise the properties or formal model (or both) if the fault lies there. This is indicated by the arrows from “Formal Verification” back to “Logical Properties” and “Formal Model.” Similarly, we may wish to gain more confidence in the correctness of the formal model and logical properties by employing one of the other V&V techniques, before proceeding to modify the real system and the other assets.

The same requirements, implemented as assertions, could then be monitored during simulation-based testing, providing another set of assurances. This technique is indicated by the arrows from “Simulator” and “Assertions” to “Simulation-based Testing.” During testing we may find requirement violations, as we did with formal verification, and we have to decide a course of action: revising the relevant assets (e.g. the simulator model, the expression of the assertions) or the robotic

system itself if we deem the violation to be genuine, or proceeding to compare the results with experiments to gain more confidence, if results were similar to formal verification. On the other hand, assurances generated by the simulation may not align with assurances generated by the other verification techniques. There are a number of potential causes of such disagreements:

- System model inaccuracies. All the verification techniques use models of the real-world. The models might have been constructed erroneously, or may be inconsistent with the real world, or relative to one another.
- Requirement model inaccuracies. In our approach, the real-world requirements of the system are converted into textual requirements, assertions and properties for verification. These requirements models may not have been correctly formulated.
- Tool inaccuracies. It is possible that numerical approximations affect the verification results. In addition, third party tools can contain bugs that are unknown to us.

We could now proceed to perform “Experiments.” As before, we may find a problem with the textual requirements or the physical system during experimentation. At the same time, the assurances from formal verification and/or simulation-based testing can be compared against the experiment results. We may also discover that one of the assurances holds during simulation-based testing or formal verification, but not during the experiments. In this case we may need to refine any of the other assets, as explained before.

Careful comparisons must be made between the different representations in order to discover the cause of the assurance conflicts. Such comparisons are indicated by the bi-directional arrows between “Formal Verification” and “Simulation-based Testing”, “Simulation-based Testing” and “Experiments”, and “Formal Verification” and “Experiments”, respectively, in Figure 1.

4 The BERT Handover Task: A Case Study

In this section, we present a case study to demonstrate the application of assurance-based verification to an HRI scenario considering the following research question: can assurance-based verification provide a higher degree of confidence in the resulting assurances than when using verification techniques in isolation?

BERT 2 is an upper-body humanoid robot designed to facilitate research into complex human-robot interactions, including verbal and non-verbal communication, such as gaze and physical gestures (Lenz et al., 2010) (see Figure 2). BERT 2’s software architecture was originally developed using YARP⁵. More recently, this system has been wrapped with a ROS interface.

⁵<http://www.yarp.it>



Figure 2: BERT 2 engaged in the handover task.

We verify an object *handover* to exemplify our approach, in the context of a broader collaborative manufacture scenario where BERT 2 and a person work together to assemble a table (Lenz et al., 2012). In the handover, the first step is an activation signal from the human to the robot. BERT 2 then picks up a nearby object, and holds it out to the human. The robot announces that it is ready to handover. The human responds verbally to indicate that they are ready to receive. (For practical reasons, human-to-robot verbal signals were relayed to the robot by a human operator pressing a key.) Then, the human is expected to pull gently on the object while looking at it. BERT 2 then calculates three binary sensor conditions:

- Gaze: The human’s head position and orientation relative to the object are tracked using the Vicon[®] motion-tracking system for an approximate measure of whether he/she is looking at the object.
- Pressure: Changes in the robot’s finger positions are sensed to detect whether the human is applying pressure to take the weight of the object.
- Location: The Vicon[®] motion-tracking system is used to determine whether the human’s hand is located on the object.

The sensor conditions must be calculated within a time threshold for BERT 2 to determine if the human “is ready”. The robot should release its grip on the object if all three conditions are satisfied. Otherwise, the robot should terminate the

handover and not release the object. The human may disengage and the robot can timeout, which would cancel the remainder of the handover task. The sensors are not completely accurate and may sometimes give incorrect readings.

A safety requirement ensures that “nothing bad happens”, whereas a liveness requirement ensures that “something good happens eventually” or inside a threshold of time for practical reasons (e.g., in simulation). The requirements for any HRI task depend on its safety and functional context. For example, in our case study the robot would need to achieve a particular handover success rate threshold to keep up with manufacturing throughput or avoid unacceptable damage costs, as per the ultimate user’s requirements. We considered two different thresholds for our first functional requirement, based on estimates of acceptable productivity in two different settings. The first threshold is considered for deployed use in a hypothetical manufacturing environment.

Req. 1a: At least 95% of handover attempts should be completed successfully.

In a research and development environment, a lower threshold may be considered satisfactory to provide proof-of-concept, showing that the system works most of the time.

Req. 1b: At least 60% of handover attempts should be completed successfully.

The following requirements were chosen to illustrate our approach, inspired by (Grigore et al., 2011) and drawing from the standards ISO 10218 (2011) for industrial robots, ISO 13482 (2014) for personal care robots, and ISO 15066 (2016) for collaborative robots:

Req. 2: If the human is not ready, the robot shall not hand over the object.

Req. 3: If the human is ready, the robot shall hand over the object.

Req. 4: The robot always reaches a decision within a threshold of time.

Req. 5: The robot shall always either time out, decide to release the object, or decide not to release the object.

Req. 6: The robot shall not close its hand when the human is too close.

Req. 7: The robot shall start in restricted speed.

Req. 8: If the robot is within 10 cm of the human, the robot’s hand speed is less than 250 mm/s.

These requirements are ambiguous in terms of how they are assessed over the available system information, reflecting the generality of the standards and the shortfalls of using natural language when first establishing requirements. In order to verify and validate them, we need to interpret them in terms of available variables and system behaviours according to the assets.

5 Assurance-Based Verification for the Case Study

After establishing the system’s requirements to verify, we developed a plan for the application of assurance-based verification to the case study. We chose to focus on a “typical use case” for the handover task, in which the human has a working familiarity with the robot and intends to complete the task successfully. Assurances based on any of the requirements may be used as bases for comparison between methods used, provided that the requirement may be modelled at all levels of abstraction. We chose to focus on the assurance results for our principal functional requirement **Reqs. 1a–b** (about the handover success rate), as a first basis for failure finding and to refine the assets if necessary. The handover success rate could be expected to be sensitive to a wide range of foreseen and unforeseen events. As a scalar measure, it allows assurances from the V&V techniques to be compared in a quantitative manner, whereas comparisons of True/False results may be insensitive to important modelling discrepancies.

After focusing on Requirement **Reqs. 1a–b**, we would proceed to verify the remaining requirements (**Reqs. 2–8**), identifying any further need to improve assets or the design itself. The verification of the full set of requirements provides a more comprehensive evaluation of the system’s requirement satisfaction, whilst facilitating the evaluation of the benefits of combining individual verification techniques to complement one another.

As mentioned previously in Section 3.3, assurance-based verification will be carried out in a reactive manner, according to the resulting assurances. In terms of the order in which we applied the V&V techniques, we chose to begin with a comparison of formal verification and simulation-based testing for requirement **Reqs. 1a–b**, to acquire as much insight as possible into the system and our modelling assumptions before committing resources to more expensive physical experiments. The subsequent stages of verification and asset modification, explained later in Section 6, were conducted with the aim of achieving a credible and trustworthy assurance of handover success rate (requirement **Reqs. 1a–b**) that was supported by all three verification techniques.

In order to apply our approach to the BERT 2 handover scenario, it was necessary to implement each element in Figure 1. Appropriate tools for formal verification and simulation-based testing were selected first. Requirements models were then translated from the textual requirements in Section 4, and system models were constructed to reflect the physical system. We developed relevant assets, for a chosen set of tools comprising the probabilistic model checker PRISM, ROS–Gazebo and a CDV testbench for simulation-based testing, and experiment designs at the BRL. We detail the development of these components in the following subsections.

5.1 Formal Verification

We chose PRISM, a probabilistic symbolic model checker (Kwiatkowska et al., 2011), for the formal verification component. In PRISM, probabilistic systems can

be modelled as discrete- and continuous-time Markov chains, Markov decision processes and Probabilistic Timed Automata (PTA). In PRISM models, transitions between states can be annotated with probabilities. The models consist of a set of modules, each representing a different process within the system being modelled. Modules are executed concurrently. Each module consists of a number of variables along with transition rules for updating those variables according to preconditions. Communication between modules is made possible by reading globally-accessible variables and by synchronisations between transitions in different modules. Execution of a PRISM model starts from an initial state (of which there can be many), and advances by application of transitions whose preconditions have been satisfied. These transitions then update the state of the model. This continues until a fixed point is reached when it is no longer possible to update the state (Parker, 2016).

Properties to verify can be expressed in a probabilistic logic such as probabilistic CTL (PCTL). Rather than outputting a Boolean value, PRISM can be used to output a probability that a given property holds for some sequence of states, or *path*, through a model (Parker, 2016).

PRISM has been used to model and verify a range of probabilistic systems such as security protocols (Duflet et al., 2013), biological systems (Konur and Gheorghe, 2015), robots and multi-robot systems have been modelled and verified (Konur et al., 2012; Llarena and Rosenblueth, 2012).

5.1.1 Formal Model

The PRISM model of the handover task consists of nine different modules: the *human*, the human’s *gaze*, *hand pressure* and *location*, the *robot* (representing BERT 2), BERT 2’s *gaze*, *pressure* and *location sensors*, as well as a *timekeeping* module which keeps track of time elapsed in the model. The model consists of around 300 lines of PRISM code and is therefore too long to reproduce in this paper. However, for illustrative purposes the robot and human modules are shown in Figures 3 and 4. The PRISM code can be interpreted as follows. The first line in the Human module defines the module. The second line defines a module variable, “humanState”, which is an integer in the range 0 to 99. Its initial value is set to “start” which is the name of a constant integer set outside the module:

```
const int start          = 0;
```

Lines 3–7 are transition rules, which determine how the state of the Human module changes over time. For example, the first rule (see line 3) says that if the human is in the state called “start”, then the state is updated to “activatedRobot.” In other words, the first thing the human does in this scenario is to activate the robot for the handover task. The rule also contains a *synchronisation* label, “activateRobot,” which means that this transition must occur at the same time as all other transitions with the same label. In this case, the only other module containing this label is the Timekeeping module, as the synchronisations in this model are used primarily

Figure 3: The Human module written in PRISM.

```

module human
  humanState : [0..99] init start;
  [activateRobot] humanState=start -> (humanState'=activatedRobot);
  [tick] humanState=activatedRobot -> (humanState'=activatedRobot);
  [informHumanOfHandoverStart] humanState=activatedRobot ->
    (humanState'=responding);
  [humanIsReady] humanState=responding -> (humanState'=setGPL);
  [tick] humanState=setGPL -> pDisengages: (humanState'=offTask) +
    pStaysOnTask: (humanState'=setGPL);
endmodule

```

Figure 4: The Robot module written in PRISM.

```

module robot
  robotState: [1100..1199] init waiting;
  handContents : [0..1000] init nothing;
  testedTimeout: bool init false;
  GPLWasOk: bool init false;
  [activateRobot] robotState=waiting -> (robotState'=
    moveHandToObjectLocation);
  [movingHand] robotState=moveHandToObjectLocation -> (robotState'=
    graspObject) & (handContents'=leg);
  [graspingObject] robotState=graspObject -> (robotState'=
    moveHandToHandoverLocation);
  [informHumanOfHandoverStart] robotState=moveHandToHandover-
    Location -> (robotState'=informedHumanOfHandoverStart);
  [humanIsReady] robotState=informedHumanOfHandoverStart ->
    (robotState'=waitForGPLUpdate);
  [GPLOkSet] robotState=waitForGPLUpdate & humanState=setGPL &
    gazeSensorState=gazeOk & pressureSensorState=pressureOk &
    locationSensorState=locationOk ->
    (robotState'=GPLOk) & (GPLWasOk'=true);
  [tick] robotState=waitForGPLUpdate & humanState=setGPL &
    (gazeSensorState=gazeNotOk | pressureSensorState=pressureNotOk |
    locationSensorState=locationNotOk) -> (robotState'=waitForGPLUpdate);
  [tick] robotState=waitForGPLUpdate & (humanState=tired |
    humanState=bored | humanState=offTask) -> (robotState'=
    interactionDone);
  [tick] robotState=GPLOk -> (handContents'=nothing) &
    (robotState'=handoverSuccessful);
endmodule

```

to keep track of how much time has elapsed. Another feature of PRISM is probabilistic non-determinism, which can be seen in lines 8–9 of the Human module, in which the human may disengage from the handover task with a probability set by “pDisengages”, or remain engaged with a probability set by “pStaysOnTask”. These are modelled as two constant double-precision floating point numbers:

```
const double pDisengages = 0;
const double pStaysOnTask = 1-pDisengages;
```

For our case study, the probability that the human disengages (i.e., becomes bored or distracted) is set to zero as we are primarily focused on the typical use-case in which the human is always focused on the task. Similarly, we assume that the human’s gaze, hand pressure and location are always within acceptable bounds for the handover task.

Real-world sensors do not work perfectly, and this is reflected in the formal model. As a result, it is possible that the handover task will not always complete successfully. The gaze sensor reports that the human is looking at the object only 95% of the time. The rest of the time the sensor reports (incorrectly) that the human is not looking at the object. When the gaze sensor reports correctly that the human’s gaze is okay, the gaze sensor has reported a “true positive.” When the gaze sensor incorrectly reports that the human’s gaze is not okay, we call this a “false negative.” Similarly, the gaze sensor may correctly report that the person is not looking at the object (a true negative, with probability 95% also) or may incorrectly report that the person is looking at the object (a false positive). Note that true positives and their corresponding false negatives are mutually exclusive, and therefore $P(\text{false_negative}) = 1 - P(\text{true_positive})$. The same is also true of true negatives and false positives:

```
const double pGazeTP = 0.95;
const double pGazeFN = 1-pGazeTP;
const double pGazeTN = 0.95;
const double pGazeFP = 1-pGazeTN;
```

The pressure and location sensors are given the same probabilities of 95% for true positives/negatives and 5% for false negatives/positives. With no experimental results or hardware specifications to refer to, it was assumed that sensors would be accurate “most of the time.” A reliability of 95% was therefore chosen as a first estimate.

5.1.2 Logical Properties

Logical properties were expressed in terms of PCTL. We use the following PCTL symbols (Parker, 2016): $\neg p$ meaning that p is not true, $p \wedge q$ meaning that both p and q are true, $p \vee q$ meaning p and/or q is true, $p \implies q$ meaning that if p is true then q is true, $F p$ meaning that eventually p will be true, $G p$ meaning that p is always true from now on, $X p$ meaning that p is true in the next state and $p \cup q$

meaning that p is true until q is true. $P(q)$ denotes the probability of q being true in the initial state.

For example, consider the requirement that “once the human is ready, BERT 2 will hand over the object.” This requirement can be implemented as a temporal logical formula:

$$G(\text{robotState=GPOk} \implies F \text{robotState=handoverSuccessful}) , \quad (1)$$

which reads “it is always the case that if gaze, position and location are correct, then eventually the handover is successful (i.e., the object is released to the human).” We can then find the probability of this formula being true on any given path through the state space. We do this by forming a property which can be analysed using a probabilistic model checker like PRISM:

$$P^{=?} (G(\text{robotState=GPOk} \implies F \text{robotState=handoverSuccessful})) \quad (2)$$

Using the operation $P^{=?}(f)$ tells the model checker that we want to find out the probability of the formula f .

Another possible requirement is that the probability of completion of the handover task should be greater than 90%. This can be rephrased as, “the success rate of the handover task is at least 90%.” This can be formulated as a property in PRISM as follows:

$$P (F \text{robotState=handoverSuccessful}) \geq 0.9 \quad (3)$$

This property states that the probability that the robot will eventually release the object is at least 0.9, or 90%.

Note that the translation of textual requirements into logical properties is not direct, since there might be different interpretations depending on the available variables, probabilities, and so on. Hence, this translation process carries the potential for misinterpretation. For example, in the properties above, “handoverSuccessful” is used as a synonym for “object handed over,” which may not be correct in all cases (e.g., the human may drop the object after release).

The full code for the PRISM models and properties used in this paper can be found online (Webster et al., 2016a).

5.2 Simulation-Based Testing

A simulator for the handover task was implemented in the ROS framework for robot code development and the Gazebo simulator. Among Gazebo’s features are support for 3D graphics rendering and various physics engines (including ODE⁶, used in this paper). Although now available as a standalone Ubuntu package, Gazebo was originally developed as a ROS package and retains its compatibility with ROS. A URDF (Universal Robot Description Format) file, used in ROS to describe the kinematic structure of the robot, actuators, and sensors, can simply be

⁶<http://www.ode.org>

extended to describe parameters used by the physics engine such as inertial properties and friction coefficients. This compatibility allows the same control code to be used in simulation and in the actual robot, providing consistency between simulations, experiments, and deployed use. A screenshot of the ROS/Gazebo simulation can be seen in Figure 5.

For the simulator, additional ROS nodes were constructed in Python, to simulate BERT 2’s sensor systems and embedded actuation controllers. The pre-existing URDF file describing BERT 2 was extended as described previously for use in Gazebo. The simulated human behaviour was controlled by a ROS node written in Python, driving a simplified physical model of the head and hand.

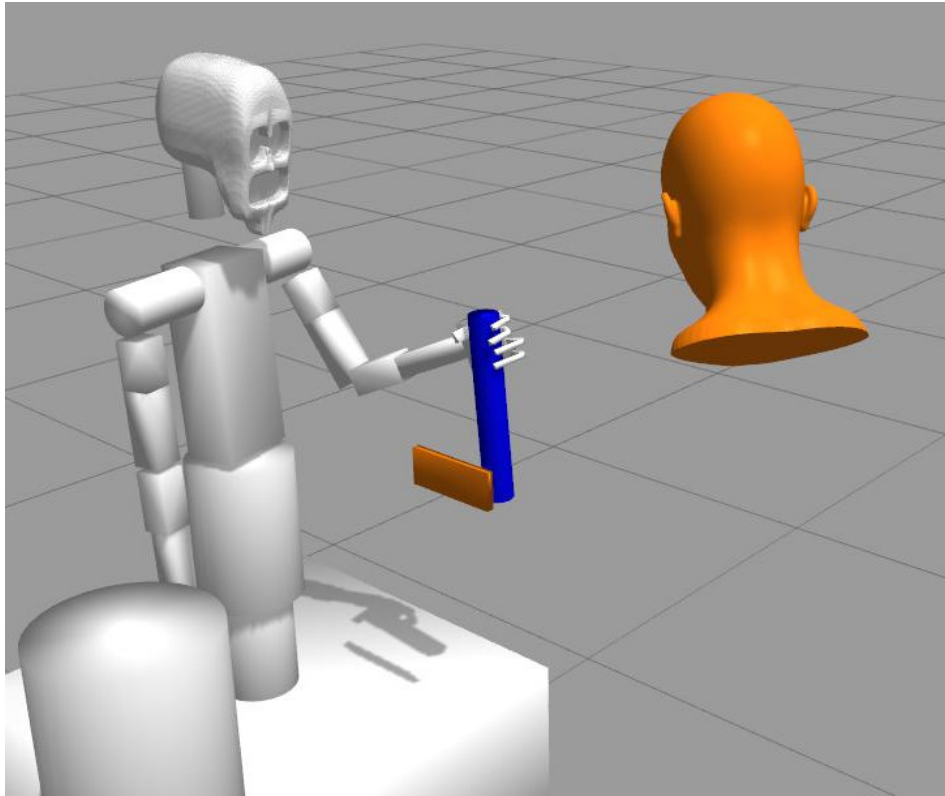


Figure 5: Screenshot of the simulated handover task. The human head and hand are represented in orange. The object to be handed over is shown in blue.

A testbench was incorporated into the simulator. The testbench comprised a test generator, a driver, a checker and a coverage collector. Achieving the exploration of meaningful and interesting sequences of behaviours from the robot and its environment in an HRI task is a challenging task. For this reason, we stimulate the robot’s code in the simulation indirectly through stimulating its environment (e.g., the person’s behaviour) instead, and we use a combination of model-based and pseudorandom test generation. Also, to alleviate the complexity of generating

and timing different types of system inputs, the test generator is based on a two-tiered approach (Araiza-Illan et al., 2016) where an abstract test is generated first and then concretized by instantiating low-level parameters. The high-level actions of the human in the simulator include sending signals to the robot, or setting abstract parameters for gaze, location and pressure. Low-level parameters include the robot’s initial pose and the poses and force vectors applied by the human during the interaction. For example, we computed an abstract test of high-level actions for the human, by exploring the model in UPPAAL⁷, so that the robot was activated (sending a signal to activate the robot and wait for the robot to present the object), the gaze, pressure and location sensor readings were correct (set gaze, pressure and location to mean “ready”), and the robot released the object. This allowed testing the requirement “if the human is ready, BERT 2 should hand over the object”.

The driver distributed the test components in the simulator. A self-checker — i.e., automated assertion monitors — was added according to the requirements, described in more detail in the following subsection. Finally, a coverage collector gathered statistics on the triggering of the assertion monitors.

The simulator code is available online⁸.

5.2.1 Assertion Monitors

For requirements checking, assertion monitors were implemented as state machines in Python, allowing sequences of events to be captured. If the precondition of an assertion is satisfied, the machine transitions to check the relevant postconditions, to determine if the assertion holds, or not. Otherwise, the postconditions are never checked.

For example, requirements **Reqs. 1a-b** and **Req. 3** were both initially monitored as the following sequence:

```
if (sensors_OK)
    wait_for(robot_decision)
    assert(robot_released_object)
```

Note that, as with the logical properties, there may be different ways to implement an assertion for the same textual requirement, and there is scope for misinterpretation.

The results of the assertion checks, if triggered, are collected and a conclusion about the satisfaction of the verified requirements can be drawn at the end of simulation. The number of times each assertion monitor has been triggered in a set of tests can be used as a measure of the coverage achieved by that test set.

⁷<http://www.uppaal.org>

⁸https://github.com/robosafe/testbench_ABV

5.3 Experiments

5.3.1 Experiment Design

BERT 2 can be verified experimentally with respect to the textual requirements using a custom facility at the Bristol Robotics Laboratory, as shown in Figure 2. When seeking to verify probabilistic properties of a system, the experiments should ideally provide an unbiased sampling, representative of the system’s deployed environment. However, some phenomena may be difficult to reproduce naturally in experiments, due to their rarity, safety considerations, or other practical limitations. Consequently, experiment-based estimates of their likelihood may be inaccurate, as may estimates of dependent properties such as the overall success rate of the task.

In the case of the handover task, we cannot confidently seek an overall success rate that accounts for the full possible range of conditions relating to hardware, software, the environment and the human (including mood, anatomy, and level of understanding of the task). Human factors are particularly challenging to test in an unbiased way. This problem can be ameliorated by acknowledging the constraints of the experiments or proactively constraining them, to achieve a more reliable characterisation of a subset of the system’s state space. The constraints become a part of the resulting assurance. Thus, the experiments deliver an estimate of “success rate within some set of constraints”, instead of an estimate of “overall success rate”. More affordable or coverable verification tools such as simulation or formal modelling may be employed to gain confidence beyond this constrained assurance.

As we were focusing on the “typical use case” of the handover scenario, in which the human has a working familiarity with the robot and intends to complete the task successfully, experiments were constrained accordingly. Subjects were given clear instructions on how to successfully complete the task, followed by practice sessions which ended when the task was successfully completed three times in a row. Subjects were instructed to try to complete the task successfully in each test. All subjects confirmed that they had no physical disability that would affect their interaction with the robot. The robot started each test in a random pose. The object was placed in a fixed location, with random orientation about its vertical axis (thus changing the orientation of the optical markers, potentially affecting sensing of the object or influencing human hand placement on grasping).

Approval for experiments with volunteer subjects was obtained from the University of the West of England’s Ethics Committee, before they took place. A large, diverse cohort enables a more comprehensive verification to be carried out, but a cohort of ten was deemed sufficient for the purpose of demonstrating our assurance-based verification approach. We recruited ten adults from the Bristol Robotics Laboratory and the local area. Most had prior robotics experience: three were post-graduate robotics students, one was a robotics entrepreneur, four were post-doctoral roboticists. Two had no prior experience of robotics. All subjects signed a consent form prior to participation.

5.3.2 Textual Requirements

For physical experiments, requirements **Reqs. 1–3** can be verified in their textual form based on visual observation, informed by video recordings and user feedback as necessary, e.g. to judge whether the human was ready or whether something has gone wrong.

Requirements **Reqs. 4–8** refer to software or physical parameters that cannot be reliably monitored by visual observation. It is therefore appropriate to implement objective monitoring to inform judgements as to whether the textual requirements are satisfied. To this end, ROS’s built-in *rosvbag* package was used to record all sensor readings, actuation signals, robot poses, and high-level control messages sent during each test. Offline monitoring of these requirements was achieved by playing back the recordings while running assertion monitors as described in Section 5.2.1.

In the case of requirements **Reqs. 6–8**, these monitors depended on the robot’s own sensing systems as the best available estimates of speed and spatial relationships. In real-world V&V exercises, independent sensing should be used.

Requirements **Req. 4** and **Req. 5** refer to the runtime behaviour of the robot’s high-level control code. Hence the monitors used in simulation may also be applied to the experiment recordings, because the same robot code is used in each case.

All experiment recordings, along with the assertion monitor reports from simulations and experiments, are available from the University of Bristol’s Research Data Repository (Webster et al., 2016b).

6 Assurance-based Verification of Requirements 1a and 1b

After generating assets for verification through different V&V techniques, we can generate assurances about the handover scenario, according to the plan described in Section 5.

We compare assurances on the handover success rate (**Reqs. 1a-b**) from formal verification (**A1**) and simulation-based testing (**A2**). Sources of discrepancy are identified and investigated in experiments with the physical system. An experiment-based assurance of the handover success rate in the ‘typical use case’ (**A3**) is then generated. More detailed system characteristics measured during these experiments are used to inform modifications to the simulator, leading to a new assurance (**A4**) that agrees closely with **A3**. These simulations also reveal a new aspect of the system’s behaviour. All insights gained up to this point are then used to inform modifications to the formal model, and the resulting assurance (**A5**) is found to agree closely with **A3** and **A4**, satisfying our objective of achieving 3-way agreement between the methods. The enacted workflow, depicted in Figure 6, is described in detail in the subsequent subsections.

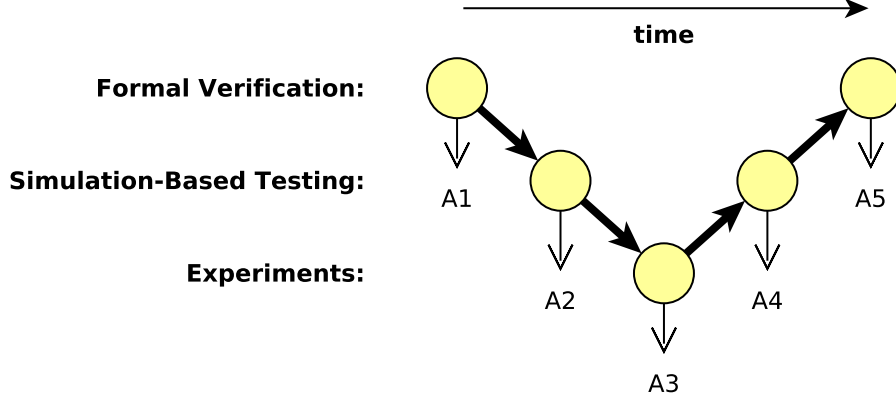


Figure 6: A simplified representation of the workflow enacted in our case study, denoting the sequence in which assurances **A1–A5** were produced from individual verification techniques.

6.1 Formal Verification: Assurance A1

As described in Section 5.1.1, the formal model includes probabilities of certain events coming to pass. Using the probabilistic model of the handover scenario we are able to determine that handover has close to 100% success rate:

$$P (\text{F handoverSuccessful}) = 0.9999948082592586 \quad (4)$$

That is, almost 100.0% of the time the handover task completes successfully. This is a very high probability of success, meaning that there are very few paths through the model which result in failure of the handover task. There are two reasons for this. First, the model is based on a typical use case (see Section 5) in which the human’s gaze, hand pressure and location are assumed to be correct at all times. This reduces the likelihood of handover failure. Second, the robot waits for all of its sensors to report that gaze, pressure and location are correct before releasing its gripper. If any of these sensors does not report an acceptable value, then the robot continues to wait. This continues until the modelled robot eventually “times out” after 100 seconds. Given that the human always responds correctly in this version of the model, and there are no other sources of unreliability in the model, the only way the model can fail is if the robot times out while waiting for the sensors to report that the human’s gaze, pressure and location are within acceptable bounds. As there are far many more paths through the model in which the handover completes successfully, the probability of success is very close to 100.0%.

The formal model has shown that BERT 2 satisfies requirements **Req. 1a** and **Req. 1b**:

Req. 1a: At least 95% of handover attempts should be completed successfully.

Req. 1b: At least 60% of handover attempts should be completed successfully.

However it is important to note that the formal model is using very rough estimates of the sensor reliabilities. To improve the accuracy of the formal model it is necessary to find more accurate figures for the sensor reliability. These could be obtained from manufacturer specifications, or through experiments with the BERT 2 robot.

Despite the shortcomings of the formal model in its current form, we can still form an assurance, which we call **A1**:

A1: the success rate of handover is 100.0%.

6.2 Simulation: Assurance A2 disagrees with A1

Assurance **A1** can now be verified by another verification technique. In this case we use simulation as it is less costly than experimentation.

Visual inspection of preliminary simulations indicated that the object sometimes fell from the robot’s hand upon grasping or during carrying (“grip failure”), a possibility not previously considered. A new assertion monitor was constructed to capture this event in isolation. Additionally, the monitor for **Req. 1** was adapted to the form below to account for the possibility of grip failure. Compared with the initial implementation presented in Section 5.2, an earlier precondition is used to trigger the monitor: (`robot_grasps_object`). The original precondition (`sensors_OK`) is now asserted as a postcondition, and is preceded by an additional postcondition (`object_contacts_robot_hand`) is asserted repeatedly until sensing is complete. This ensures that a verdict of ‘False’ will be returned if the robot drops the object prematurely, regardless of any subsequent behaviour.

```
if (robot_grasps_object)
    while !sensing_Done
        assert(object_contacts_robot_hand)
        assert(sensors_OK)
        wait_for(robot_decision)
        assert(robot_released_object)
```

In a set of 100 simulations of the handover task, 80 attempts then were completed successfully. This result forms assurance **A2**:

A2: the success rate of handover is 80%.

Note that **A1** and **A2** disagree with each other. This highlights the benefits of the assurance-based approach to verification: we can use different verification techniques to confirm or deny assurances. As explained in Section 3.3, there are a number of potential causes of such a disagreement: inaccuracies in either the system models or the requirement models, or in the tools. The latter becomes more unlikely when established tools are used.

In our case, the occurrence of grip failure was clearly the main source of discrepancy; a modelling inaccuracy was present in at least one of the two verification techniques used. The formal model implicitly assumed a grip failure rate of 0%,

Table 1: User experiment results, by subject. Robotics experience is denoted by N (none), S (post-graduate student), D (post-doctoral roboticist), or E (robotics entrepreneur). Failure modes are denoted by R (robot grip failure), P (false negative pressure sensing), and L (false negative location sensing).

Subject ID	1	2	3	4	5	6	7	8	9	10	mean
Experience	E	D	S	N	N	D	S	D	D	S	
# training runs	5	4	6	6	6	3	3	6	3	6	4.8
# successes (post-training)	9	10	8	7	9	10	10	6	9	10	8.8
Failure modes	P		P, L	R, P, P	R			L, P, L, P	P		

whereas simulation indicated 20%. Before committing resources to user experiments, a set of hardware experiments was conducted to characterise the actual robot’s grip failure rate. BERT 2 was programmed to carry out the grasp-and-carry portion of the handover task 100 times, and grip failure was found to occur in 3 cases.

At this point, despite some discrepancy, formal modelling and simulation were in agreement that the system satisfied the research-level minimum success rate of 60%. Furthermore, the simulation-based estimate was deemed likely to be conservatively low, due to the inaccurately high grip failure rate. It was therefore deemed worthwhile to proceed to user experiments.

6.3 Experiments: Assurance A3

User experiments were carried out as described in Section 5.3. Results are summarized in Table 1. To determine whether it was appropriate to treat our experimental data as independent samples of a single distribution, we investigated whether there was any noticeable effect of learning or prior robotics experience on the outcome of a test. A statistical analysis was performed using IBM® SPSS® v23.0. A Kruskal-Wallis H Test did not indicate a significant effect of the robotics experience categories on the number of handovers completed successfully after training ($\chi^2(3) = 1.5$, $p = 0.682$), or on the number of training runs required ($\chi^2(3) = 2.872$, $p = 0.412$). Furthermore, a Spearman correlation revealed no significant correlation between the test number (1–10) and the total number of human-related failures in that test across subjects ($\rho = 0.220$, $p = 0.541$, two-tailed). It is possible that more extensive testing with a larger cohort would reveal weak but statistically significant effects of these parameters. However, for the purposes of demonstrating our method, our cohort and experiment design was deemed to be adequate based on these results.

The handover was successfully completed in 88 out of 100 tests. As in simulation, this can be taken as an estimate of the true success rate of the experimental system.

A3: the success rate of handover is 88% in the typical use case.

Here the ‘typical use case’ is that described in Section 5.3.

Again we found notable disagreement between **A3** and the previously generated assurances. A more specific discrepancy had already been identified in terms of the grip failure rate. To seek closer agreement between the three verification techniques, we explored the potential sources of discrepancy in greater detail.

The video recordings and ROS logs, including sensor data, were reviewed to confirm the faults responsible for each failed handover in the user experiments. The failure rate for each failure mode was identified as the number of occurrences divided by the number of opportunities for that fault to occur. Results are listed in the first column of Table 2. ‘False negative’ here was defined relative to the subject’s observable actions. Thus false negative pressure sensing was identified where the review of logs and videos indicated that the subject was observably applying pressure to the object but the sensing threshold was not exceeded. Similarly, false negative location sensing was identified where the subject’s hand was on the object during the sensing period but the robot’s location sensor returned a negative result. Rates of other possible failure modes (e.g. timeouts or false negative gaze sensing) are implicitly estimated to be 0% based on these experiments. This should not be taken as evidence that these modes never occur, only that they are rare. Also, rates of false positive sensor readings could not be defined because, after training, there were no cases in which the subject did not apply their gaze, pressure, and hand location according to the protocol.

6.4 Modifying the Simulator

The observed rates for individual failure modes were taken as the best available estimates of those properties in the typical use case, and were used to tune the simulator (and, subsequently, the formal model) to represent that case.

In the previous simulations, the grip failure rate of 20% was clearly much higher than the experimental observation of 3%, while the simulated sensing did not reproduce the other observed failure modes. Several aspects of the simulator were refined with the aim of approximating the experimentally observed rates of individual failure modes without sacrificing realism.

The accuracy of the simulated dynamics of the robot’s handling of the object was improved by replacing default/placeholder values with more realistic estimates of inertial properties, material properties, and joint torque/velocity limits.

The instances of false negative ‘location’ sensing were identified as arising from the motion tracking system briefly losing track of the object (hand location is measured relative to the object) and reassigning its location to another point. Mimicking this behaviour, the simulated motion tracking was set to reassign the observed location of the object (but not the person’s hand or head) to an arbitrary point in 3.1 percent of readings.

Based on the recordings, all cases of false negative pressure sensing seen in the user experiments were attributed to the subject pulling on the object more gently than in other cases. The exact forcing pattern applied by the subjects could not be

extracted from the experiment data. Instead, the lower threshold of the distribution from which the simulated human pulling force was selected was lowered from 5 N to 1 N through a process of trial-and-error to approximate the failure rate seen in user experiments.

After tuning, a set of 500 simulations was run. In all tests the simulated human enacted the trace of high-level actions corresponding to the typical use case, remaining engaged in the task and applying their gaze, pressure, and location within the relevant bounds. The results, included in Table 2, indicate that the tuning process was successful in approximating the individual failure rates observed in user experiments. Close correspondence is also achieved in the handover success rate, although it must be acknowledged that this correspondence slightly overestimates the true accuracy of the simulator; larger errors are seen in the rates of individual failure modes. Nevertheless, we have improved confidence in the simulation as a representation of the physical system, and in the assurances provided by each method. **A3** is now supported by a new assurance from simulation-based testing:

A4: The success rate of handover is 87.8% in the typical use case.

Table 2: Test outcomes and occurrence rates of individual failure modes for the typical use case, according to user experiments and simulations after tuning.

	User experiments	Simulation
Number of tests	100	500
Handover success	88.0% (88/100)	87.8% (439/500)
Grip failure	2.0% (2/100)	1.6% (8/499)
False negative gaze	0.0% (0/98)	0.0% (0/491)
False negative pressure	7.1% (7/98)	6.5% (32/491)
False negative location	3.1% (3/98)	4.2% (21/491)
Runtime error	0.0% (0/100)	0.2% (1/500)

Furthermore, the simulations exposed a failure mode not previously considered. In one test, the handover success monitor returned no result and inspection of the logs revealed that the robot’s control code crashed due to a runtime error:

```
RuntimeError: Unable to connect to
    move_group action server 'place'
    within allotted time (2)
```

This message indicates that a timeout occurred when invoking the robot’s motion planning module. The robot’s high-level control code does not include any means of handling such exceptions. Although rare, these events may significantly affect the user’s trust if they occur in deployment, and could lead to violations of critical safety requirements. In our case, the error caused the only violations of **Reqs. 4–5**. The exposure of the error, which required high-volume testing and a realistic implementation of the system, demonstrates a key strength of simulation

as a complement to formal modelling and user experiments. It is conceivable that the error never occurs in the actual system, e.g. due to differences in computational load during simulation. However, further testing on the real system cannot rule the possibility out completely. A more conservative approach is to adopt the simulation-based estimate of the error’s frequency as the basis for further assurances and design recommendations.

6.5 Modifying the Formal Model

Now that we have verified assurance **A3** in simulation, we can attempt to re-verify it using formal verification to address the discrepancy discovered between **A1** and **A3** during the first V&V cycle. As described in Section 6.1, assurance **A1** generated by formal verification disagrees with assurance **A3**, generated by experiments:

A1: the success rate of handover is 100.0%.

A3: the success rate of handover is 88% in the typical use case.

The formal model currently uses placeholder estimates for the reliability of the gaze, pressure and location sensors on the BERT 2 robot. However, using some of the experimental data in Table 2 it is possible to replace the corresponding estimates in the formal model with more accurate values. In particular, we can use the following values:

- Gaze sensor, false negative: 0.0%
- Pressure sensor, false negative: 7.1%
- Location sensor, false negative: 3.1%

False negatives and true positives are mutually exclusive, since the former refers to when the person’s gaze/pressure/location is correct but the sensor reports (incorrectly) that it is not, and the latter refers to when the person’s gaze/pressure/location is correct and the sensor reports (correctly) that it is. Therefore, we can infer true positive values:

- Gaze sensor, false negative: 0.0%, true positive: 100.0%
- Pressure sensor, false negative: 7.1%, true positive: 92.9%
- Location sensor, false negative: 3.1%, true positive: 96.9%

As the experiments did not report any situations where there were false positives, we assume that the rate of false positive sensor failures is 0.0% for each sensor, and therefore the rate of true negatives for each sensor is 100.0%.

We can now set the probabilities in the model accordingly:

```

const double pGazeFN          = 0.00;
const double pGazeTP          = 1-pGazeFN;
const double pGazeFP          = 0.00;
const double pGazeTN          = 1-pGazeFP;
const double pPressureFN      = 0.071428571;
const double pPressureTP      = 1-pPressureFN;
const double pPressureFP      = 0.00;
const double pPressureTN      = 1-pPressureFP;
const double pLocationFN      = 0.030612245;
const double pLocationTP      = 1-pLocationFN;
const double pLocationFP      = 0.00;
const double pLocationTN      = 1-pLocationFP;

```

Verifying the model we can obtain the success rate of the handover task:

$$P (\text{F robotState=handoverSuccessful}) = 0.9999954384256133 \quad (5)$$

It can be seen that the success rate remains at almost 100.0%. This is to be expected as the sensor failure rates have changed slightly, but it remains the case that the only way for the handover to fail is for the robot to time-out.

There is still a significant difference between this success rate and the success rate reported by simulation (87.8%) and experiments (88%). This may be, in part, due to the way in which the sensors were modelled in the formal model. It was assumed that sensors might make any number of “samples” while the robot is waiting for the person to grasp the object in the correct way. Each one of these samples is a separate event, in which the sensor takes a reading which is reported back to the robot’s decision-making system. Therefore, each time the sensor takes a reading there is a probability of failure, and false positives and negatives are possible. The formal model reflects this, and the failure rates above apply to each reading taken by the sensor, rather than the average failure rate per handover. The PRISM code defining the gaze sensor module was as follows:

```

module gazeSensor
  gazeSensorState : [0..1000] init null;

  [senseGaze] robotState=waitForGPLUpdate &
    gazeState=gazeOk -> pGazeFN: (gazeSensorState'=
      gazeNotOk) + pGazeTP: (gazeSensorState'=gazeOk);

  [senseGaze] robotState=waitForGPLUpdate &
    gazeState=gazeNotOk -> pGazeTN: (gazeSensorState'=
      gazeNotOk) + pGazeFP: (gazeSensorState'=gazeOk);
endmodule

```

The first transition rule says that if the robot is currently waiting for the person to grasp the object (“waitForGPLUpdate”) and the gaze is okay (i.e., the person is looking in the right direction), then update the value of “gazeSensorState” to either “gazeOk” or “gazeNotOk”, depending on the probability of false negative and true positive. The second transition rule does something similar for the case where the person is not looking in the right direction. Note that the only guards on these transitions specify that the robot is waiting for the person to grasp the object and that the gaze is either okay or not okay. (The synchronisation “senseGaze”

is simply used to keep track of how long sensing is taking within a timekeeper module, and is not relevant in this example.) Therefore, these sensor readings can happen any number of times while the robot is waiting for the person to be ready to receive the object and complete the handover task.

This way of modelling the handover scenario produces less accurate results when combined with the failure rates established by experiment. This is because the failure rates determined were based on the number of experiments in which, for example, the pressure sensor was seen to give a false negative reading. For example, the probability of 0.071 for a pressure sensor false negative reading was obtained by dividing the number of experiments in which a false negative reading occurred at some point (7) by the total number of experiments not interrupted by gripper failure (98).

Therefore it would be more accurate to re-model the scenario in a way that reflects experimental reality; that is, the probability of a sensor failure for a handover of the object should be based on the observed average rate of failure of that sensor. This was achieved by modifying the gaze, pressure and location sensor modules in the PRISM model:

```
module gazeSensor
  gazeSensorState : [0..1000] init null;

  gazeSensorSet: bool init false;
  [senseGaze] robotState=waitForGPLUpdate &
    gazeState=gazeOk & !gazeSensorSet ->
    pGazeFN: (gazeSensorState'=gazeNotOk) &
      (gazeSensorSet'=true) + pGazeTP:
      (gazeSensorState'=gazeOk) & (gazeSensorSet'=true);

  [senseGaze] robotState=waitForGPLUpdate &
    gazeState=gazeNotOk & !gazeSensorSet ->
    pGazeTN: (gazeSensorState'=gazeNotOk) &
      (gazeSensorSet'=true) + pGazeFP:
      (gazeSensorState'=gazeOk) & (gazeSensorSet'=true);
endmodule
```

In this revised model each sensor's state can be set only once. For example, for the gaze sensor, this is done by introducing a Boolean variable "gazeSensorSet" which is initially false, but is set to true once a sensor reading has been taken, and is never again set to false. Therefore, this model reflects the experiments more closely.

Verifying the new model gives us a new value for the reliability of the handover task:

$$P (F_{\text{robotState=handoverSuccessful}}) = 0.9001457729154516 \quad (6)$$

The handover task now completes successfully with a probability of 90.0%. This is closer to the simulation and experiment results of 87.8% and 88.0% respectively, but there is still a noticeable difference. One possible reason for this is that the gripper failure rate, as determined by experiment and built into the simulation, is not yet modelled in PRISM. The following transition describes what happens within the robot module once the gaze, pressure and location are found to be correct:


```
[tick] robotState=GPLOk -> (handContents'=nothing) &
(robotState'=handoverSuccessful);
```

Here, once the robot's state reaches "GPLOk," indicating that gaze, pressure and location are within acceptable bounds, the robot releases its gripper and hands over the object to the person. Therefore the "handContents" variable is updated to reflect that the robot's hand/gripper is now empty, and the robot's state is updated to show that handover has been successful. To introduce the possibility of gripper failure, this transition was modified to incorporate a probabilistic choice:

```
[tick] robotState=GPLOk -> pGripperOk: (handContents'=
nothing) & (robotState'=handoverSuccessful) +
pGripperFailure: (handContents'=nothing) &
(robotState'=handoverUnsuccessful);
```

Now, one of two things can happen. The first possibility is that the handover completes successfully, as before, with a probability of "pGripperOk." The second is that the handover fails, with probability "pGripperFailure." These two probabilities are set like so:

```
const double pGripperFailure = 0.02;
const double pGripperOk = 1 - pGripperFailure;
```

Here, "pGripperFailure" is set to 0.02 in accordance with the gripper failure rate of 2% determined by experiment (see Table 2). We verify the model once again to determine a handover success rate of 88.2%:

$$P(\text{robotState}=\text{handoverSuccessful}) = 0.8821428574571426 \quad (7)$$

In a similar way a new transition was introduced into the transition system to model the possibility of failure of BERT 2's motion planning module, as described in Section 6.4. This transition occurs at the start of the handover task as the robot prepares to move its arm to grasp the object for handover. The revised transition rule incorporates probabilities for the success or failure of the motion planning module:

```
[activateRobot] robotState=waiting -> pMotionOk:
(robotState'= moveHandToObjectLocation) +
pMotionFailure: (robotState'=motionError);
```

These probabilities were based on the data shown in Table 2:

```
const double pMotionFailure = 0.002; // 0.2
const double pMotionOk = 1 - pMotionFailure;
```

Verifying the model once more gives an updated handover success rate of 88.0%:

$$P(\text{robotState}=\text{handoverSuccessful}) = 0.8803785717422283 \quad (8)$$

This success rate is much closer to the simulation (87.8%) and experimental results (88.0%). In an attempt to improve accuracy even further the model was closely

examined and compared again to the experimental set-up and simulation, but no possible further improvements were found. Thus the final assurance provided by formal verification may be stated as:

A5: the success rate of handover is 88.0% in the typical use case.

After conducting an assurance-based verification of the handover task for the BERT 2 system, it was found that all verification techniques were in close agreement on the probability of a successful handover. The probabilities are shown in Table 3.

Table 3: Results of assurance-based verification.

Formal Verification	88.0%
Simulation	87.8%
Experiments	88.0%
Average	$87.9\% \pm 0.1\%$

Having established confidence in our models, we can give credible assurance that, in the typical use case, **Req. 1b** is satisfied but **Req. 1a** is not.

7 Verification of Requirements 2–8

In order to demonstrate the assurance-based approach for verification of a robotic system, we focused our efforts in the previous section on the verification of **Reqs. 1a–b**. However, for the sake of completeness, and in line with best practices in engineering, we also attempted verification of **Reqs. 2–8** using each of the three verification techniques. These verification results are presented without reference to assurances, but the assurance-based approach to verification could be applied to **Reqs. 2–8** in a similar manner to **Reqs. 1a–b**.

It can be seen in the following sub-sections that the different verification techniques do not all agree on how well **Reqs. 2–8** are met. This is similar to the case study for **Reqs. 1a–b** before the application of the assurance-based approach to verification. Given enough time, it would be possible to apply the assurance-based approach to this expanded set of requirements in order to find the source of the disagreements between verification techniques and to improve the level of agreement between them.

7.1 Experiments

For the user experiments, the full set of textual requirements was evaluated through a combination of offline assertion monitoring and visual observation, as described in Section 5.3.2. Table 4 presents the verdicts returned from each individual test. **Req. 1** is included for completeness. Note that, for requirements **Reqs. 4–8**, up to seven of the missing verdicts were attributable to errors in the recording process, rather than the tests themselves.

Table 4: User experiments: Results on textual requirements from 100 tests. “Covered” indicates the number of tests from which a verdict could be achieved. “Passed” and “Failed” indicate the number of tests in which the requirement was deemed to be satisfied or violated, respectively. “Pass rate” is calculated as the ratio “Passed”：“Covered”.

Req.	1	2	3	4	5	6	7	8
Covered	100	0	98	93	93	25	98	90
Passed	88	0	88	93	93	25	78	90
Failed	12	0	10	0	0	0	20	0
Pass rate	0.88	—	0.90	1.0	1.0	1.0	0.8	1.0

As noted previously, the handover success rate in the user experiments satisfies **Req. 1b** but violates **Req. 1a**. Correspondingly, violations of **Req. 3** arise from the cases of false negative sensor readings. Additionally, we see that **Req. 7** is violated in 78 out of 98 tests; the robot occasionally violates its speed threshold upon resetting, presumably depending on its initial pose. A notable “coverage hole” is seen in this test set for **Req. 2**, as the human was judged to be ready for the handover in every test. All other requirements were covered in at least 25 tests, and no other violations were observed.

7.2 Simulation-based Testing

Table 5 presents the results of the assertions monitored in the same 500 simulation-based tests summarised in Table 2, representing the typical use case. Comparing Table 5 with the experiment results in Table 4, we see broad agreement, but with several noteworthy discrepancies discussed below.

Table 5: Simulation: assertion coverage and results corresponding to each of the requirements (corrected for missing results), in a set of 500 tests.

Req.	1	2	3	4	5	6	7	8
Covered	500	53	446	500	500	500	500	0
Passed	439	53	446	499	499	500	437	0
Failed	61	0	0	1	1	0	63	0
Pass rate	0.878	1.0	1.0	0.998	0.998	1.0	0.874	—

All assertions were covered — i.e. all monitors were triggered at least once — except for **Req. 8**. This indicates that the human and robot did not come within 10 cm of each other during the interaction. While this is possible given the length of the object to be handed over, the experiments revealed that closer proximities are seen in typical use. Hence this constitutes a notable coverage hole in these tests.

Contrary to the experiment results, **Req. 2** was covered in several tests and no violations of **Req. 3** were observed. Further investigation of this discrepancy revealed a potential requirements inaccuracy; the assertion corresponding to this requirement expressed “the human is ready” as `sensors_ok`. In this sense, the

assertion monitor verifies only the high-level control of the robot, discounting the possibility of sensor errors. Hence the results are still informative, but some modification of the assertion monitors would be required to achieve a more comprehensive verification of these requirements.

As noted previously, **Req. 4–5** were violated by the single runtime error. The observation that **Req. 7** is violated in 63 out of 500 tests is consistent with the experiment results.

7.3 Formal Verification

Req. 2 says that if the human is not ready, the robot shall not hand over the object. It was formalised as follows:

$$P \left(G \left[\neg \left(\begin{array}{l} \text{gazeState} = \text{gazeOk} \wedge \\ \text{pressureState} = \text{pressureOk} \wedge \\ \text{locationState} = \text{locationOk} \end{array} \right) \Rightarrow \left(\begin{array}{l} \text{robotState} = \text{handoverSuccessful} \vee \\ \text{robotState} = \text{handoverUnsuccessful} \end{array} \right) \right] \right) \quad (9)$$

This property says that it is always the case that if the human’s gaze, pressure and hand location are not correct, then it is not the case that the robot has attempted to hand over the object. (Handing over the object results in either the “handoverSuccessful” or “handoverUnsuccessful” states.) Verifying this property in PRISM gives a probability of 1.0, meaning that it is always true.

Req. 3, which says that, “if the human is ready, the robot shall hand over the object,” was formalised in a similar way:

$$P \left(G \left[\left(\begin{array}{l} \text{gazeState} = \text{gazeOk} \wedge \\ \text{pressureState} = \text{pressureOk} \wedge \\ \text{locationState} = \text{locationOk} \end{array} \right) \Rightarrow F \left(\text{robotState} = \text{handoverSuccessful} \right) \right] \right) \quad (10)$$

It was expected that this property would be evaluated by PRISM as less than 1.0 due to the possibility of sensor and gripper failures. Indeed, verification using PRISM gave a result of 0.8803785717422283.

Req. 4 states that the robot always reaches a decision within a threshold of time. This was formalised as follows:

$$P \left(G \left[\left(\text{robotState} = \text{GPLOk} \right) \Rightarrow F \left(\left(\begin{array}{l} \text{robotState} = \text{handoverSuccessful} \vee \\ \text{robotState} = \text{handoverUnsuccessful} \end{array} \right) \wedge \text{robotState} = \text{objectReleaseTimer} \leq 20 \right) \right] \right) \quad (11)$$

Here the phrase “reaches a decision” was taken to mean that the robot has decided to release the object. In the model, this can result in “handoverSuccessful” if the gripper works properly, or “handoverUnsuccessful” if the gripper fails. The requirement specifies that this should happen within “a threshold of time,” but does not specify the amount of time. In our model, we specified that the gripper release would take 2.0 seconds based on consultation with the robot’s users. Time was

quantified in the model using a “objectReleaseTimer” which is set to zero when the robot determines that the humans’ gaze, pressure and location are acceptable. The objectReleaseTimer was set to work in 0.1 second intervals in order to provide adequate precision without increasing the size of the state space to intractable levels. Therefore the property above captures **Req. 4** as it states that once the robot has found the human’s gaze, pressure and location to be acceptable, then it will attempt to release the gripper (either successfully or unsuccessfully) within 2.0 seconds.

This property was verified and the probability was determined to be 0.9999999-999999996, or 100.0%, allowing for floating point arithmetic precision errors in PRISM’s computation engine (Parker, 2016).

Req. 5 states that the robot shall always either time-out, decide to release the object, or decide not to release the object. It was formalised as follows:

$$P \left(G \left[\begin{array}{l} (F \text{ robotState} = \text{handoverSuccessful}) \vee \\ (F \text{ robotState} = \text{handoverUnsuccessful}) \vee \\ (F \text{ robotState} = \text{waitForGPLUpdate} \cup \text{robotState} = \text{timedOut}) \end{array} \right] \right) \quad (12)$$

This property specifies the probability that it is always the case that the robot eventually decides to release the object (either successfully or unsuccessfully), or times out while waiting for the human’s gaze, pressure and location to update to acceptable values. The latter case, where the robot times out, is effectively the same as the robot deciding to not hand over the object.

This property was verified revealing a probability of 0.9979999999999996, which was expected as the invocation of the motion planning module has a failure rate of 0.002. In order to check that this was the case another property was specified which includes the error relating to the motion planning module:

$$P \left(G \left[\begin{array}{l} (F \text{ robotState} = \text{handoverSuccessful}) \vee \\ (F \text{ robotState} = \text{handoverUnsuccessful}) \vee \\ (F (\text{robotState} = \text{waitForGPLUpdate} \cup \text{robotState} = \text{timedOut})) \vee \\ (F \text{ robotState} = \text{motionError}) \end{array} \right] \right) \quad (13)$$

This property was verified resulting in a probability of 0.9999999999999993, or 100.0% allowing for precision errors.

Req. 6 states that the robot shall not close its hand when the human is too close. This was formalised as follows:

$$P \left(G \left[\begin{array}{l} \text{robotState} = \text{moveHandToObjectLocation} \wedge \\ \text{proximityState} = \text{proximityNotOk} \\ \implies \\ \neg X (\text{robotState} = \text{graspObject}) \end{array} \right] \right) \quad (14)$$

This property says that it is always the case that if the robot is in state “moveHandToObjectLocation” and the proximity of the human is not acceptable (i.e., the human is too close) then it is never the case that in the next state the robot updates its state to “graspObject.” This captures the property as “moveHandToObjectLocation” is the state which precedes “graspObject,” when the robot has moved its gripper to pick up the object prior to handover, but has not yet grasped it.

This property was verified and the probability was found to be 0.92514843-00669699. It was expected that the probability would be in the range $0 < P < 1$ as the robot’s current design does not include a proximity sensor and therefore on some of the paths through the system the robot will satisfy the property, and on others it will not.

Req. 7 says that the robot shall start in a restricted speed mode. This limits the robot’s speed to less than 250 mm/s. This property could not be verified formally as the robot’s current design does not allow for different speed modes to be set within the control system, so it would be incorrect to include such a feature within the formal model.

Req. 8 says that if the robot is within 10 cm of the human, the robot’s hand speed is less than 250 mm/s. This can be formalised as follows: As was the case with **Req. 7**, this property cannot be verified formally as the model includes information on whether the human is “too close” or not, and does not currently contain more detailed information about the precise distance from the human to the robot. It is possible, in principle, to re-model the scenario to include such detail. However, adding complexity to the model adds to the computational resources required to verify the model. In some cases, verification can become intractable. Therefore, it may be more practical for **Reqs. 6–8** to rely more heavily on assurances gained from simulation and experiment where physical properties can be much more fine-grained.

7.4 Computational Demands

Properties 4 to 14 were verified against several different PRISM models representing the handover task. The complexity of the PRISM model checking for these properties is shown in Table 6. From left to right, the columns show the property verified, numbers of states and transitions used, memory required for model construction, time required for building the model and time required to verify the model. (The memory used for Property 9 is not shown as this property was proven trivially by the PRISM verification engine.) PRISM 4.2.1 was used on an eight-core Intel® Core™ i7 laptop with 16 GB of memory running Ubuntu Linux 12.04.

It can be seen that Properties 10–14 took significantly longer to verify than the other properties. This is most likely the result of the use of nested temporal logic operators (e.g., F, G, X, U) in these properties compared to Properties 4–8, which use simpler formulae. Properties 9–13 took the same time to build the model (69.4 seconds) as these properties were all checked against a single PRISM model file, which needed to be built only once before these properties could be verified.

Simulation-based testing was performed using ROS Indigo and Gazebo v2.2.3 on a quad-core Intel® Core™ i7 laptop with 8 GB of memory running Ubuntu Linux 14.04. With all online monitors running, simulations were executed at a speed of $0.8 \times$ real-time on average, taking 69.3 seconds per test. Considering the advantages of batch processing, simulation-based testing is still considerably faster than physical experiments.

Table 6: Complexity of formal verification using PRISM.

Prop.	States	Transitions	Build	Verification	
			Time (s)	Time (s)	Memory (kB)
4	42,960	236,643	36.8	0.203	2,253
5	31,120	150,955	61.4	0.147	1,741
6	15,614	54,969	84.0	0.062	997
7	15,615	54,971	90.0	0.057	999
8	15,623	54,998	40.6	0.053	1,003
9	15,623	54,998	69.4	0.002	—
10	15,623	54,998	69.4	20.3	1,536
11	15,623	54,998	69.4	17.7	1,007
12	15,623	54,998	69.4	24.9	1,843
13	15,623	54,998	69.4	50.2	2,048
14	46,869	227,498	106.8	92.5	3,379

8 Discussion

Through the assurance-based combination of formal verification, simulation-based testing, and experiments, we have delivered an assurance of handover success rate (**Reqs. 1a–b**) with greater confidence than could be achieved by any of the V&V techniques in isolation. Although the experiments alone would have returned a similar value for the handover success rate, achieving corroboration in model checking, with an abstract model, gives confidence that the formal model is a correct interpretation of the HRI scenario. Furthermore, for **Reqs. 4–6**, the combination of simulation-based testing and formal verification exposed important system behaviours not observed in the experiments, i.e. requirement violations. The observed runtime error (which caused violations of **Reqs. 4–5**) could only be exposed through a high number of tests in simulation. The subsequent inclusion of this event in the formal model ensured that its impact on the behaviour of the system could be exhaustively explored, thus obtaining corrected models for two techniques for V&V to balance coverability capabilities and detail as well as realism.

Our process has confidently demonstrated that the system satisfies **Req. 1b**, but not the more stringent version, **Req. 1a**. Based on the insights gained during the verification process, several design recommendations could be made to improve the handover success rate and to satisfy other requirements. The sensing process could be made more robust to sudden changes in the human motion, or to reduce the number of handover failures due to sensing errors, through mechanisms such as “debouncing” for the sensor readings. Adjustments to the robot’s hardware or motion planning strategy may improve the gripper failure rate. A speed limit needs to be introduced when the robot is reset, to avoid dangerous unintended collisions. Also, as uncontrollable faults can be encountered during execution, we could instrument our code to perform diagnostics and fault recovery strategies.

For demonstration purposes, we focussed on achieving full agreement on assurances relating to a particular set of requirements, **Reqs. 1a–b**. As our examination of **Reqs. 2–8** demonstrates, agreement on some requirements does not entail agreement in all the requirements. For **Reqs. 1a–b**, the end result was that all verification techniques agreed on the success rate of handover within a range of $\pm 0.1\%$.

In an ideal world all verification methods are accurate models of the world, and are accurate with respect to one another, so that an assurance generated with one verification method should also be found valid by another. In practice, this may not happen. If two verification methods do not agree, then we might look for inaccuracies in the system models, the requirements models, or the tools as described in Section 3.3. However, after a number of iterations through the assurance-based verification diagram, we may still have verification methods in disagreement about assurances. One possible reason might be project constraints: we may lack the resources to continue to address inaccuracies. Another reason could be that the verification methods may be lacking: for example, model checking for formal verification can often be hindered by state space explosions which limit the accuracy of models that can be checked. Alternatively, we might lack the computational resources to explore sufficient numbers of simulated experiments, or we may lack the personnel to conduct sufficient numbers of real experiments.

Therefore, in practice, consensus between verification methods may not be possible. At this point we might assess whether our verification methods are up to the job. Perhaps we should use an automated theorem prover rather than a model checker? Or perhaps a two-dimensional physical simulation would work better than a three-dimensional one? Perhaps we could create the simulation using a different programming language, or use a more powerful computer? The list goes on.

We might also decide that consensus is not necessary if all the verification methods are within an acceptable range. For example, we might have three different assurances, each generated by a different verification method:

A_i : System reliability is 92%.

A_j : System reliability is 98%.

A_k : System reliability is 93%.

Clearly all three assurances are not in agreement with the others. However, the lowest value for system reliability given by the assurances is 92%, which means that all three verification methods agree on the following assurance:

A_l : System reliability is 92% or greater.

Note that A_l is implicit in assurances A_i , A_j and A_k . In this case we have used assurance-based verification to allow us to determine a minimum value for reliability. This value can then be checked with respect to system requirements to see whether the system being modelled is sufficiently reliable.

There may be other reasons, beyond those discussed in this section, why we cannot reach consensus on assurances between verification methods, and there may be other ways to remedy this beyond range-based assurances. It is intended that the suggestions given here may provide direction for managing assurance-based verification in practical applications, as well as acknowledging that assurance-based verification is not perfect. Rather, it is an approach to using verification methods

in conjunction with one another to provide a higher degree of confidence in the assurances that a system will satisfy its requirements.

As we have improved the accuracy of our assets on the basis of the results presented in this paper, we could use the same V&V techniques to further explore the HRI by, for example, applying human behaviours that deviate from the typical use case. A reformulation of the system and requirement models, under the new conditions, might be necessary, as system traits (e.g. failure rates) characterised under one set of constraints will not necessarily hold for other sets of constraints. The verification engineer should use their judgement in such cases as to whether any prior asset modifications can be generalised to broader scenarios.

The verification efforts towards assurance agreement can be helped by limiting (or biasing) the explored region of the HRI state space, to seek cases in which the V&V techniques provide contradictory results. In our case study, we have employed a probabilistic formulation of the requirements that is relevant to HRI systems as non-determinism may arise not only from the environment but also from the robot and the coupling between them (ROS-based robots exhibit high levels of concurrency and run on non-real-time operating systems). Hence, we can compute conditional probabilities, such as, “Given that the robot’s gripper fails, what is the probability that the robot warns the user before the object drops?,” that lead to conditional assurances.

In more complex scenarios, it may become more difficult to identify appropriate modifications to achieve better agreement between assets. Modifications to system models may be informed by knowledge gained during verification or previously. For example, useful insights may be contributed from systematic risk analyses, such as Fault Tree Analysis or HAZOP (Hazard Operability). The latter has recently been proposed for use in human-robot interactions to manage the inherent complexity and uncertainty in such systems (Guiochet et al., 2013).

Increased modelling effort is an evident limitation of assurance-based verification. However, the use of multiple verification techniques brings savings to this effort. As our case study demonstrates, early use of more abstract methods allows gradual commitment of resources to more realistic and expensive techniques. Discrepancies can highlight oversights and areas of uncertainty, informing the judicious use of more expensive techniques (e.g., to characterise the uncertain grip failure rate before proceeding with user experiments, in our case study).

For more comprehensive verification efforts, Coverage Driven Verification (Araiza-Illan et al., 2016) may be combined with assurance-based verification, pursuing coverage of the system in a systematic way, in simulations or experiments. Hybrid systems methods (Julius et al., 2007; Kim et al., 2006) might also be usefully incorporated into assurance-based verification, although reducing entire HRI scenarios to manageable hybrid models is likely to be challenging.

The object handover is only an example of a huge variety of case studies available in the HRI domain. Nonetheless, it is of uttermost interest in HRI, as close-proximity manipulation tasks may be considered in a plethora of applications, such as manufacture of white goods, cooperative handling and attachment of large sub-

components of aeroplane structures in aerospace assemblies, or care of the elderly with early stage dementia by feeding them soup. Our approach can be extended to any HRI application, in principle.

9 Conclusions

We presented an approach, denominated assurance-based verification, for the V&V of robotic assistants, to help in demonstrating their trustworthiness in the context of human-robot interactions. Our approach provides integral assurances on robot's safety and functional correctness, through the combination of formal verification (model checking), simulation-based testing, and user validation in experiments with a real robot. The use of these V&V techniques provides assurances at different degrees of *coverability* (i.e. the exploration of the HRI task) and HRI modelling detail, thus overcoming the shortfalls of each method when applied in isolation. For example, model checking provides an exhaustive exploration of a system model, at a cost on system detail lost in an abstract model. Combined with simulation-based testing, we gain high-fidelity detail by running the real software, but we cannot test the whole state space of variables and behaviours. Also, an iterative process between the three V&V techniques can be used if the resulting assurances present discrepancies, refining and improving the artefacts (i.e., system and requirement models) to represent the HRI task in a more truthful manner. This allows gaining confidence in the resulting assurances about the safety and functional correctness of the robot.

We demonstrated our assurance-based V&V approach through a handover task, a safety-critical part of a complex cooperative manufacture scenario, for which we proposed some safety and liveness requirements to verify and validate. We constructed formal models (Probabilistic Timed Automata), a simulator (in the Robot Operating System and Gazebo), and a test rig for the HRI (in the Bristol Robotics Laboratory), as well as temporal logic properties, and tests and assertion checkers from the requirements. The V&V focus starts with a pair of requirements, **Reqs. 1a–b**, for which we sought assurance agreement with the three techniques, by modifying the formal model and the simulator. We then examined the rest of the requirements, finding previously unknown functional failures in the system. Our results showcase the benefits of our approach, in terms of thorough exploration of the system under verification at different levels of detail and completeness, and in terms of gaining confidence in the V&V results.

9.1 Future Work

We will investigate how the translational potential of our proposed approach can be improved by more explicit evaluations of confidence. For example, Guiochet et al. (2015) summarise various qualitative and quantitative approaches to assessing confidence in an assurance. They present a quantitative model describing the

propagation of confidence through particular argument structures. The results of our demonstration of assurance-based verification constitute an “alternative argument” structure, in that separate pieces of evidence each support the whole assurance. Where one method supports only part of an assurance – e.g. where testing is limited to a subset of the state space to which the assurance applies, while formal modelling covers the whole state space – this may be accounted for by applying weighting factors to the individual pieces of evidence. For probabilistic traits of a system, statistical techniques such as the modified Wald method (Agresti and Coull, 1998) can be used to quantify the uncertainty (confidence intervals) arising from the limited number of tests feasible in simulation or experiment. However, it should be noted that such confidence intervals do not describe the accuracy of the models themselves. Hence the implementation of quantitative models of confidence propagation will often rely on informal estimates of the confidence in individual pieces of evidence.

Finally, we intend to apply the method to a broader collaborative manufacturing task, of which the handover may be a subcomponent.

Acknowledgments

This work is part of the EPSRC-funded project “Trustworthy Robotic Assistants” (refs. EP/K006320/1, EP/K006193/1 and EP/K006223/1).

References

- Agresti A and Coull BA (1998) Approximate is Better than Exact for Interval Estimation of Binomial Proportions. *The American Statistician* 52(2): 119–126. DOI:10.1080/00031305.1998.10480550. URL <http://dx.doi.org/10.1080/00031305.1998.10480550>.
- Aitken JM, Veres SM and Judge M (2014) Adaptation of System Configuration under the Robot Operating System. *IFAC Proceedings Volumes* 47(3): 4484–4492. DOI:10.3182/20140824-6-ZA-1003.02531. URL <http://www.sciencedirect.com/science/article/pii/S1474667016423050>.
- Alami R, Albu-Schaeffer A, Bicchi A, Bischoff R, Chatila R, Luca AD, Santis AD, Giralt G, Guiochet J, Hirzinger G, Ingrand F, Lippiello V, Mattone R, Powell D, Sen S, Siciliano B, Tonietti G and Villani L (2006) Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1–16. DOI:10.1109/IROS.2006.6936985.
- Araiza-Illan D, Western D, Pipe AG and Eder K (2015) Coverage-Driven Verification – an approach to verify code for robots that directly interact with humans.

- In: *Hardware and Software: Verification and Testing, LNCS*, volume 9434. pp. 69–84.
- Araiza-Illan D, Western D, Pipe AG and Eder K (2016) Systematic and Realistic Testing in Simulation of Control Code for Robots in Collaborative Human-Robot Interactions. In: Alboul L, Damian D and Aitken JM (eds.) *Towards Autonomous Robotic Systems*, number 9716 in Lecture Notes in Computer Science. Springer International Publishing. ISBN 978-3-319-40378-6 978-3-319-40379-3, pp. 20–32. URL http://link.springer.com/chapter/10.1007/978-3-319-40379-3_3.
- Arnold J and Alexander R (2013) Testing autonomous robot control software using procedural content generation. In: *Computer Safety, Reliability, and Security, LNCS*, volume 8153. pp. 33–44.
- Behdenna A, Dixon C and Fisher M (2009) Deductive Verification of Simple Foraging Robotic Behaviours. *International Journal of Intelligent Computing and Cybernetics* 2(4): 604–643.
- Bordini RH, Fisher M and Sierhuis M (2009) Formal verification of human-robot teamwork. In: *Proc. ACM/IEEE HRI*. pp. 267–8.
- Broenink JF, Ni Y and Groothuis MA (2010) On model-driven design of robot software using co-simulation. In: *Proc. of Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*. pp. 659–668.
- Clarke E, Grumberg O, Jha S, Lu Y and Veith H (2000) Counterexample-Guided Abstraction Refinement. In: Emerson EA and Sistla AP (eds.) *Computer Aided Verification*, number 1855 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. ISBN 978-3-540-67770-3 978-3-540-45047-4, pp. 154–169. URL http://link.springer.com/chapter/10.1007/10722167_15. DOI: 10.1007/10722167.15.
- Clarke EM, Grumberg O and Peled DA (1999) *Model Checking*. USA: MIT Press.
- Clarke EM, Klieber W, Nováček M and Zuliani P (2012) *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, chapter Model Checking and the State Explosion Problem. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-35746-6, pp. 1–30. DOI:10.1007/978-3-642-35746-6_1. URL http://dx.doi.org/10.1007/978-3-642-35746-6_1.
- Corbett JC, Dwyer MB, John H and Robby (2000) Bandera: a Source-level Interface for Model Checking Java Programs. In: *Proc. 22nd International Conference on Software Engineering (ICSE)*. ACM Press, pp. 762–765.

- Dixon C, Winfield AF, Fisher M and Zeng C (2012) Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems* 60(11): 1429 – 1441. DOI:<http://dx.doi.org/10.1016/j.robot.2012.03.003>. URL <http://www.sciencedirect.com/science/article/pii/S0921889012000474>. Towards Autonomous Robotic Systems 2011.
- Duflot M, Kwiatkowska M, Norman G, Parker D, Peyronnet S, Picaronny C and Sproston J (2013) *Formal Methods for Industrial Critical Systems: A Survey of Applications*, chapter Practical Applications of Probabilistic Model Checking to Communication Protocols. IEEE Computer Society Press, pp. 133–150.
- Eder K, Harper C and Leonards U (2014) Towards the safety of human-in-the-loop robotics: Challenges and opportunities for safety assurance of robotic co-workers. In: *Proc. IEEE ROMAN*. pp. 660–665.
- Fisher M (2011) *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley.
- Fitting M (1996) *First-Order Logic and Automated Theorem Proving*. Springer.
- Foster H, Krolnik A and Lacey D (2004) *Assertion-based Design*. 2nd edition. Springer.
- Gallardo M, Joubert C, Merino P and Sanán D (2012) A model-extraction approach to verifying concurrent C programs with CADP. *Science of Computer Programming* 77(3): 375–392.
- Gastin P and Oddoux D (2001) Fast LTL to Bchi Automata Translation. In: Berry G, Comon H and Finkel A (eds.) *Computer Aided Verification*, number 2102 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. ISBN 978-3-540-42345-4 978-3-540-44585-2, pp. 53–65. URL http://link.springer.com/chapter/10.1007/3-540-44585-4_6. DOI: 10.1007/3-540-44585-4_6.
- Grigore EC, Eder K, Lenz A, Skachek S, Pipe AG and Melhuish C (2011) Towards safe human-robot interaction. In: *Towards Autonomous Robotic Systems*. pp. 323–335.
- Guiochet J, Hoang QAD and Kaaniche M (2015) A Model for Safety Case Confidence Assessment. In: Koornneef F and Gulijk Cv (eds.) *Computer Safety, Reliability, and Security*, number 9337 in Lecture Notes in Computer Science. Springer International Publishing. ISBN 978-3-319-24254-5 978-3-319-24255-2, pp. 313–327. URL http://link.springer.com/chapter/10.1007/978-3-319-24255-2_23. DOI: 10.1007/978-3-319-24255-2_23.
- Guiochet J, Hoang QAD, Kaaniche M and Powell D (2013) Model-based safety analysis of human-robot interactions: The MIRAS walking assistance robot. In:

- 2013 *IEEE International Conference on Rehabilitation Robotics (ICORR)*. pp. 1–7. DOI:10.1109/ICORR.2013.6650433.
- Havelund K and Rosu G (2002) Synthesizing monitors for safety properties. In: *Proc. TACAS*. pp. 342–356.
- Hawkins R, Kelly T, Knight J and Graydon P (2011) A New Approach to creating Clear Safety Arguments. In: Dale C and Anderson T (eds.) *Advances in Systems Safety*. Springer London. ISBN 978-0-85729-132-5 978-0-85729-133-2, pp. 3–23. URL http://link.springer.com/chapter/10.1007/978-0-85729-133-2_1. DOI: 10.1007/978-0-85729-133-2_1.
- Hazim MY, Qu H and Veres SM (2016) Testing, Verification and Improvements of Timeliness in ROS Processes. In: Alboul L, Damian D and Aitken JM (eds.) *Towards Autonomous Robotic Systems*, number 9716 in Lecture Notes in Computer Science. Springer International Publishing. ISBN 978-3-319-40378-6 978-3-319-40379-3, pp. 146–157. URL http://link.springer.com/chapter/10.1007/978-3-319-40379-3_15. DOI: 10.1007/978-3-319-40379-3_15.
- Heitmeyer CL (2007) Formal methods for specifying, validating, and verifying requirements. *Journal of Computer Science* 13(5): 607–618.
- Huang J, Erdogan C, Zhang Y, Moore B, Luo Q, Sundaresan A and Rosu G (2014a) ROSRV: Runtime Verification for Robots. In: Bonakdarpour B and Smolka SA (eds.) *Runtime Verification*, number 8734 in Lecture Notes in Computer Science. Springer International Publishing. ISBN 978-3-319-11163-6 978-3-319-11164-3, pp. 247–254. URL http://link.springer.com/chapter/10.1007/978-3-319-11164-3_20. DOI: 10.1007/978-3-319-11164-3_20.
- Huang Z, Alexander R and Clark J (2014b) Mutation testing for jason agents. In: *Engineering Multi-Agent Systems, LNCS*, volume 8758. pp. 309–327.
- Intana A, Poppleton MR and Merrett GV (2013) Adding value to WSN simulation through formal modelling and analysis. In: *Fourth International Workshop on Software Engineering for Sensor Network Applications (SESENA)*. pp. 24–29.
- ISO 10218 (2011) ISO 10218-1:2011 robots and robotic devices – safety requirements for industrial robots – part 1: Robots.
- ISO 13482 (2014) ISO 13482:2014 robots and robotic devices — safety requirements for personal care robots.
- ISO 15066 (2016) ISO/TS 15066:2016 robots and robotic devices – collaborative robots.

- Julius AA, Fainekos GE, Anand M and I Lee GJP (2007) Robust test generation and coverage for hybrid systems. In: *Proc. HSCC*. pp. 329–342.
- Kelly T and Weaver R (2004) The Goal Structuring Notation A Safety Argument Notation. In: *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*.
- Kim J, Esposito JM and Kumar R (2006) Sampling-based algorithm for testing and validating robot controllers. *International Journal of Robotics Research* 25(12): 1257–1272.
- Kirwan R, Miller A, Porr B and Prodi PD (2013) Formal modeling of robot behavior with learning. *Neural Computation* 25(11): 2976–3019.
- Konur S, Dixon C and Fisher M (2012) Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems* 60(2): 199–213.
- Konur S and Gheorghe M (2015) A property-driven methodology for formal analysis of synthetic biology systems. *IEEE/ACM Trans Comput Biol Bioinform* 12(2): 360–371.
- Kwiatkowska M, Norman G and Parker D (2011) PRISM 4.0: Verification of probabilistic real-time systems. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, LNCS, volume 6806. Springer.
- Lenz A, Lallee S, Skachek S, Pipe A, Melhuish C and Dominey P (2012) When shared plans go wrong: From atomic- to composite actions and back. In: *Proc. IROS*. pp. 4321–4326.
- Lenz A, Skachek S, Hamann K, Steinwender J, Pipe A and Melhuish C (2010) The BERT2 infrastructure: An integrated system for the study of human-robot interaction. In: *Proc. IEEE-RAS Humanoids*. pp. 346–351.
- Llarena A and Rosenblueth DA (2012) Model checking applied to humanoid robotic soccer. In: *Advances in Autonomous Robotics*, LNCS, volume 7429. pp. 256–269.
- Lyons DM, Arkin R, Liu TM, Jiang S and Nirmal P (2013) Verifying performance for autonomous robot missions with uncertainty. In: *IFAC Symposium on Intelligent Autonomous Vehicles*. pp. 179–186.
- Meenakshi B, Bhatnagar A and Roy S (2006) Tool for translating Simulink models into input language of a model checker. In: *Formal Methods and Software Engineering*, LNCS, volume 4260. pp. 606–620.
- Mukhopadhyay D (2015) Automatic model extraction from C code– abstracter and architecture. In: *Information Systems Design and Intelligent Applications*, number 340 in AISC. pp. 389–398.

- Naks T, Olive X and Kai OSY (2009) Automatic code generator speeds development of safety-critical real-time embedded systems. *ITEA 2 Magazine* (4): 20–21.
- Nielsen B (2014) Towards a method for combined model-based testing and analysis. In: *Proc. MODELSWARD*. pp. 609–618.
- Parker D (2016) *PRISM 4.2.1 Manual*. Department of Computer Science, University of Oxford.
- Pedrocchi N, Vicentini F, Matteo M and Tosatti LM (2013) Safe human-robot cooperation in an industrial environment. *International Journal of Advanced Robotic Systems* 10(27): 1–13.
- Petters S, Thomas D, Friedmann M and von Stryk O (2008) Multilevel testing of control software for teams of autonomous mobile robots. In: *Proc. SIMPAR*.
- Pinho T, Moreira AP and Boaventura-Cunha J (2014) Framework using ROS and SimTwo simulator for realistic test of mobile robot controllers. In: *Proc. CONTROLO*. pp. 751–759.
- Piziali A (2004) *Functional verification coverage measurement and analysis*. Kluwer Academic.
- Ringert JO, Roth A, Rumpe B and Wortmann A (2014) Code generator composition for model-driven engineering of robotics component & connector systems. In: *Proc. MORSE*. pp. 66–77.
- Robinson S (1997) Simulation model verification and validation: increasing the users’ confidence. In: *Proc. Winter Simulation Conference*. pp. 53–59.
- Sargent R (2013) Verification and validation of simulation models. *Journal of Simulation* 7(1): 12–24.
- Walter D, Täubig H and Lüth C (2010) Experiences in applying formal verification in robotics. In: *Computer Safety, Reliability, and Security, LNCS*, volume 6351. pp. 347–360.
- Webster M, Cameron N, Jump M and Fisher M (2013) Generating Certification Evidence for Autonomous Unmanned Aircraft Using Model Checking and Simulation. *J. of Aerospace Information Systems*.
- Webster M, Dixon C, Fisher M, Salem M, Saunders J, Koay KL, Dautenhahn K and Saez-Pons J (2015) Toward reliable autonomous robotic assistants through formal verification: A case study. *IEEE Transactions on Human-Machine Systems* (99): 1–11. DOI:10.1109/THMS.2015.2425139.

- Webster M, Western D, Araiza-Illan D, Dixon C, Eder K, Fisher M and Pipe AG (2016a) An assurance-based approach to verification and validation of human-robot teams: PRISM code and properties for the formal verification of the BERT2 handover scenario. University of Liverpool Research Data Catalogue. <http://dx.doi.org/10.17638/datacat.liverpool.ac.uk/178>.
- Webster M, Western D, Araiza-Illan D, Dixon C, Eder K, Fisher M and Pipe AG (2016b) An assurance-based approach to verification and validation of human-robot teams – data from simulations and experiments. University of Bristol Research Data Repository. <https://data.bris.ac.uk/data/dataset/gw4qbvkmmekl1rlkgaqskdtyd>, DOI:10.5523/bris.gw4qbvkmmekl1rlkgaqskdtyd.
- Xie F, Levin V, Kurshan RP, and Browne JC (2004) Translating software designs for model checking. In: *Fundamental Approaches to Software Engineering*, volume 2984. pp. 324–338.